VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Anti DDoS systém postavený na open-source platformě

Open Source Anti DDoS System

2018

Bc. Petr Müller

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Petr Műller**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Anti DDoS systém postavený na open-source platformě
Open Source Anti DDoS System

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem diplomové práce je navrhnout otevřený systém pro zachycení a blokování útoků využívajících odepření síových služeb.

Řešení práce spočívá ve splnění následujících bodů:
1. Studium a popis síťových počítačových útoků.
2. Studium a popis technik pro zábránění síťových útoků.
3. Návrh open-source sytému pro detekci a potlačení síťových útoků
4. Testování a ověření vytvořeného systému v laboratorním prostředí.

Seznam doporučené odborné literatury:

[1] Bhattacharyya D.K., Kalita J.K.*DDoS Attacks: Evolution, Detection, Prevention, Reaction, and Tolerance.* CRC Press 2016
[2] Mirkovic J. *Internet Denial of Service: Attack and Defense Mechanisms.* Prentice Hall 2005

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Nevlud**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018

.....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2018 .....

Rád bych poděkoval vedoucímu práce Ing. Pavlu Nevludovi za odbornou pomoc, konzultaci a čas, který si našel, aby mi pomohl k vypracování této diplomové práce. Nadále rodině, blízkým a všem, kteří mi s prací pomohli.

Abstrakt

Diplomová práce se zabývá návrhem open-source systému pro detekci a potlačení síťových útoků, testování a ověření vytvořeného systému v laboratorním prostředí. V teoretické části jsou přiblíženy základní pojmy a typy síťových počítačových DoS/DDoS útoků. Dále je teoretická část zaměřená na metody obrany před síťovými útoky a nástroje používané k útokům s hodnocením aktuální situace.

Nadále jsou v práci slovně popsány nutné úkony k úspěšnému spuštění open-source programu na Ubuntu 14.04.

Samotné testování, které bylo prováděno v laboratoři školy, je doplněno o výstupy grafů z paketových analyzátorů, okomentování daných kroků a výstupů. Open-source program byl vytvořen k zautomatizování potlačení síťových počítačových útoků, ochraně zařízení a služeb. Celý kód je přiložen jako příloha na konci práce.

Klíčová slova: DoS, DDoS, C#, .NET Core, Linux, Botnet, UDP, TCP, ICMP, Hping3

Abstract

This master‘s thesis deals with the design of an open-source system for the detection and reduction of network attacks, testing and verification of a created system within a school laboratory. In the theoretical section there is a description of various basic and network DoS/DDoS attacks. Other sections will explain the principles of DoS/DDoS defensive mechanisms and the tools used for DoS/DDoS attacks with a rating of actual situational effectiveness on the field.

In my thesis you can also find the required steps for the successful use of the open-source program on Ubuntu 14.04.

The actual testing, which was done in the school’s lab, completes a presentation of graph outputs from packet analyzers and also comments on the steps. The program was created to automate the reduction of network attacks and protection of devices and services. The whole code is placed at the end of the document as an appendix.

Key Words: DoS, DDoS, C#, .NET Core, Linux, Botnet, UDP, TCP, ICMP, Hping3

Obsah

Seznam použitých zkratek a symbolů

DoS	–	Denial of service
DDoS	–	Distributed denial of service
IT	–	Information Technology
IP	–	Internet Protocol
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol
ICMP	–	Internet Control Message Protocol
SYN	–	Synchronize
ACK	–	Acknowledge
OS	–	Operation System
ARP	–	Address Resolution Protocol
DNS	–	Domain Name System
MAC	–	Medium Access Control
GUI	–	Graphical User Interface
API	–	Application Programming Interfaces
RAM	–	Random Access Memory
ARPANET	–	Advanced Research Projects Agency NETwork
HOIC	–	High Orbit Ion Cannon
LOIC	–	Low Orbit Ion Cannon
C&C	–	Command & Control
IRC	–	Internet Relay Chat
HTTP	–	Hypertext Transfer Protocol
HTTPS	–	Hypertext Transfer Protocol Secure
P2P	–	Peer-to-peer
SMS	–	Short Message Service
IoT	–	Internet of Things
NTP	–	Network Time Protocol
CPU	–	Central Processing Unit
TLS	–	Transport Layer Security
SMTP	–	Simple Mail Transfer Protocol
ACL	–	Access Control List
IPS	–	Intrusion Prevention Systems
IDS	–	Intrusion Detection System
SIEM	–	Security Information and Event Management
ISP	–	Internet Service Provider

Seznam obrázků

1 Úvod

Cílem této diplomové práce je navrhnout otevřený systém pro zachycení a blokování útoků využívajících odepření síťových služeb. Tato práce je rozvinuta do několika částí a také si klade za cíl přiblížit čtenářům problematiku DoS/DDoS útoků a s ním úzce spjatým tématem botnet sítí.

Začátek práce bude věnován základním pojmům v oblasti útoků, které vedou k odepření síťových služeb, rozdělení, historii a vysvětlení rozdílů mezi nimi. Navážu na problematiku botnet sítí, které v současnosti díky velkému rozmachu IoT tvoří obrovskou útočnou sílu v arzenálu útočníků. Další kapitolou práce bude popsání dnes nejčastěji používaných typů síťových počítačových DoS/DDoS útoků, na kterou bude navazovat další kapitola týkající se metod obrany před síťovými útoky. V této části si zmíníme zejména systémy, které se používají v praxi pro zamezení síťovým útokům.

Následující kapitola se bude týkat nejčastějších nástrojů pro samotný útok. Bude kladen důraz na nástroje, které se dají po krátké instalaci hned používat, a uživatel pro jejich použití nepotřebuje hlubší znalost funkčnosti těchto nástrojů. Některé z nich budou použity i v této diplomové práci. V této kapitole taky ve zkratce zmíním, co vede útočníky k samotným útokům a také bude zmíněna aktuální situace.

Pak už bude přistoupeno k realizaci praktické části, která bude testována v laboratoři školy. Rozdělená bude na dvě části, první části je simulace útoku bez ochrany koncového zařízení, kde budou názorně předvedeny možnosti útočícího nástroje a zařízení v laboratoři. Útočit budu pomocí tří protokolů UDP, ICMP a TCP. Útok bude popsán a zaznamenán v paketovém analyzátoru Wireshark.

Závěrečná část práce bude vyhrazena Anti-DDoS programu, který bude spuštěn na operačním systému Ubuntu 14.04 pomocí MonoDevelop. Program bude napsán v jazyce C# .NET Core, aby plnil pravý význam open-source systému a předcházel tak problémům při spuštění na jiných operačních systémech. Tento systém bude opět ověřen v laboratoři školy s uvedením topologie, výstupy z paketových analyzátorů spuštěném na tomtéž stroji jako program, pro ověření funkčnosti programu a výsledky práce. Tento program bude přiložen k práci jako příloha.

Poslední kapitolou práce bude závěr, kde shrnu diplomovou práci, práci v laboratoři a dosažené výsledky vytvořeného programu, popřípadě komplikace, které se při tvorbě programu a jeho spuštěním v laboratoři školy objeví.

2 Základní pojmy

Útoky typu DoS a DDoS jsou dnes již velice běžnými. Mezi útoky zaujímají jednu z nejčastějších forem útoků vůbec. Tyto útoky jsou velice často vedeny proti internetovým službám nebo webovým stránkám. Nejčastější typ DoS nebo DDoS útoku je zahlcení služby, kdy se běžnému uživateli stránka, na kterou přistupuje, bude zdát jako nedostupná, nebo v tom lepším případě jen zpomalená.

Dalším typem, se kterým se při útoků setkáváme u útoku odepření služby, je využití chyb v protokolu, podrobněji se na to v práci zaměříme později. Pokud stránka vykazuje jednu z příznaků odmítnutí služby, neznamená to automaticky, že se o útok odepření služby jedná.

2.1 Denial of Service

Častěji se můžeme setkat jen se zkratkou **DoS**, který v českém překladu znamená odepření služby. Tento útok jde pouze z jednoho zařízení. Často se ale můžeme setkat s tím, že touto zkratkou DoS se souhrnně označují všechny typy útoků Denial of Service.[4]

Hlavní myšlenkou a cílem je odepřít legitimním uživatelům přístup na jednotlivé služby, kteří pak nemohou pokračovat ve své práci. Existuje mnoho cílů útoku, například se jedná o útok na šířku pásma, procesor, vlastnosti protokolů, nebo napadení samotné logiky služby, aby nemohl plnit pravidelně svou službu.[4]

Motivace útočníků bývá různorodá, od osobních důvodů po politické důvody, nebo útoky vedené jen tak pro zábavu až po finanční důvody, více v sekci Motivace DDoS útoků. Někdy se jedná přímo o uživatele, který je napaden, ne o samotnou službu.

DoS se snaží znepřístupnit služby a to už jakýmkoli způsobem. Dá se říct, že to je od vytržení kabelu ze zařízení až po využití slabiny v aplikaci, v podstatě cokoliv, co zapříčiní znepřístupnění služby ostatním uživatelům.[4]

2.2 Distributed Denial of Service

Tento název se skrývá pod označením **DDoS**, dnes již velice používaný druh útoků, zejména díky velkému rozmachu botnet sítí. Útok je veden z více počítačů najednou. Velice častým úkazem je podvrhování IP adres a znemožnění vystopovatelnosti útočníka. U DDoS útoků se můžeme velice často setkat se záplavovými útoky. [1]

Útok DDoS je charakterizován mnoha agenty, kteří jsou koordinování za jediným, účelem a to útokem na cílový systém prostřednictvím sítě. Agenti jsou obvykle součástí botnetu. DDoS útoky jsou šířeny jako koordinované lidské útoky na počítačovou sít pomocí nástrojů jako HOIC, LOIC, GoldenEye a další, více pak v sekci Nástroje pro útoky.[1]

Rozlišujeme tři kategorie DDoS založených na jeji cíli:

- Volumetrické

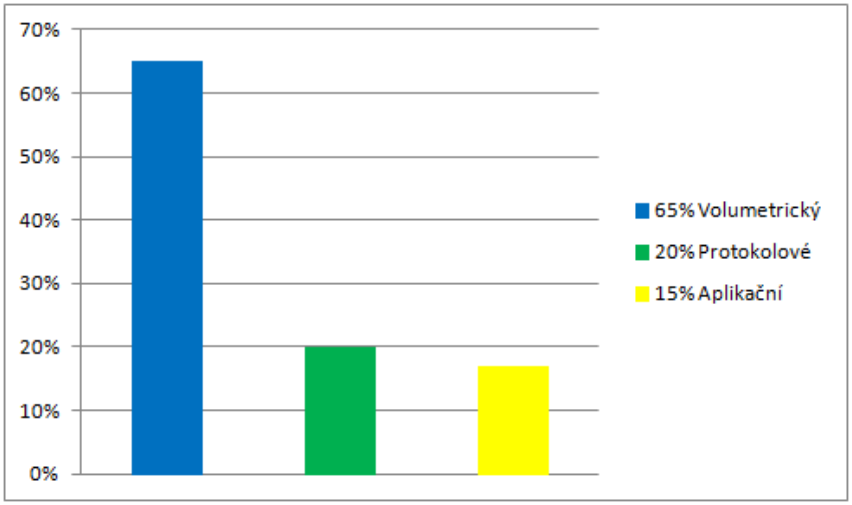
- Protokolové
- Aplikační

Volumetrické útočí na kapacitu sítě, zahlcením paketů, čímž brání legitimním uživatelům k připojení.

Protokolové jsou náchylné k útokům, které využívají vlastnosti protokolu samotného, jako jsou doba čekání, nebo tabulka připojení. Útoky jsou většinou směřovány na služby, jako například webový server. Tyto typy útoků se zaměřují na zranitelnost systémů, které způsobují, že jsou vyčerpány systémové prostředky jako RAM, místa na disku, dosažení limitu síťové karty, využití aplikace a další.

Aplikační útok naopak od volumetrických útoků nemusí být z hlediska objemu dat významný. Obvykle se útočí na některou ze známých zranitelností cílového prostředí, jejíž zneužití vede k pádu aplikačního serveru nebo jeho přetížení.

Útoky na aplikaci jsou vybírány hned z několika příčin. Směřují na aplikace a protokoly, které zahrnují atributy příhodné pro odmítnutí služby. Způsoby jsou velice sofistikované a při útoku není potřeba tolika útočníků jako například při útoku na síťovou infrastrukturu. Další výhodou tohoto útoku je fakt, že útok prakticky splývá z normálního provozu a je proto těžké ho odhalit. Výsledkem je pak opět nedostupná služba.



Obrázek 1: Procentuální vyjádření využití typů DDoS útoků, zdroj:[19]

Rozdíl mezi DoS a DDoS se nachází pouze v tom, kolik strojů je v útoku zapojeno. Při DoS útoku je zapojen jeden stroj, tedy jeden jedinec. DDoS útokům říkáme těm, při kterých je zapojeno dva a více strojů, jde o organizovaný útok. V této práci se dále budeme zabývat hlavně DDoS útoky, které jsou svou podstatou věci velice jednoduché a mají za úkol taky jednoduchý a často dosažitelný cíl, znepřístupnění služeb. Útoky DDoS patří k velice nepříjemným, velice špatně se proti němu brání.[1]

2.3 Distributed Reflection Denial of Service

S tímto pojmem, označovaný pod zkratkou **DRDoS**, se setkáváme již méně. Obecně se více mluví o útocích DoS a DDoS, avšak tento útok vychází přímo z DDoS útoku a liší se hlavně tím, že přístroje, které útočí, nejsou napadená. Většinou dojde přes botnet síť zaslání příkazu a jako zdrojovou adresu podvrhne adresu oběti. Zařízení pak na podvrženou adresu odpovídají. Výhodou tohoto útoku je jeho síla. Útočník i s velice malým počtem počítačů dokáže vyvinout masivní útok.[1]

3 Historie DoS a DDoS, Botnet síť

3.1 Historie DoS útoků

Už v dobách ARPANETu se vyskytly nedostatky v protokolech a tím se i vyskytla pravděpodobnost k útoku DoS. Již v roce 1985 se pan Andrew D. Birrell zmínil o riziku úmyslného odmítnutí služby ve článku[9].

Vzhledem k tomu, že ARPANET využívali především odborníci, tak i přesto se tady vyskytly případy, kdy došlo k útoku typu odmítnutí služby. Dokonce v roce 1988 jedna taková událost dovedla síť ARPANET na jeho kolena. Dne 2. listopadu roku 1988, doktor Robert Morris vyvinul počítačového červa, jehož cílem bylo poukázat na zranitelnost celé sítě. Červ po svém úspěšném spuštění zhroutil několik počítačů a po celé zemi narušil celou síť. Náklady na vyčištění a odstranění tohoto červa z celé sítě a počítačů se následně vyšplhaly až do výše tisíců dolarů v přímé závislosti na míře poškození napadeného stroje. Doktor Morris byl v důsledku toho odsouzen za počítačový podvod.

3.2 Historie DDoS útoků

První střet s tímto typem útoku odepření služby se pokládá incident z roku 1999, kdy výsledek způsobil dvoudenní výpadek sítě na univerzitě v Minnesotě. Nadcházející rok se tento druh útoků dostal do podvědomí širší veřejnosti po útocích na světoznámé internetové portály, jako je CNN, Amazon, Yahoo. Tyto portály byly zasaženy záplavovými útoky. Výsledek byl efektivní a na několik hodin tyto portály nebyly přístupné.

Tímto úspěšným útokem se rozpoutaly další série útoků, které byly vedeny stejným stylem, jako ten z roku 2000. Další útoky poté následovaly na internetové služby, jako je například DNS a další, které způsobily velké potíže uživatelům Internetu.

Dnes se distribuovaný útok typu Denial of Service provádí v několika krocích. Prvním krokem je získání botnetu, nebo agentů, které může útočník ovládat. Botnety jde získat na černém trhu, nebo deep webu, za peníze, nejčastějším typem měny je potom Bitcoin. V další fázi útočník přistupuje k C&C serveru, serveru botnetů a posílá příkaz agentům, aby začali odesílat určitý typ dat do cílového počítače.

Techniky IP spoofingu se používají ke krytí stop pro agenty a zejména pro C&C server. Po dosažení cíle útoku přestanou agenti odesílat pakety a služba se vrátí do normálního provozu.

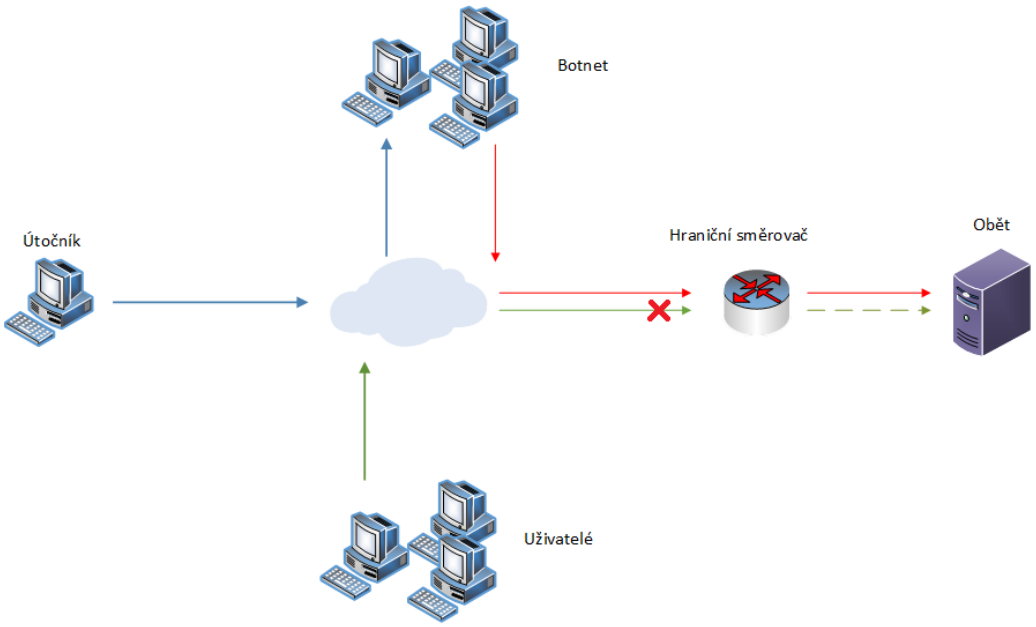
3.3 Botnet síť

Tuto síť tvoří skupina počítačů, které jsou nakažené malwarem. Většina uživatelů, kteří jsou napadeni malwarem o tomto viru neví a vědět ani nemusí, tento vir se ani žádným způsobem v počítači neprojevuje. Vir v počítači čeká až ho Bootmaster probudí ze spánku a z počítače uživatele se stane útočící nástroj, který bude po síti posílat velké množství žádosti na útočníkem specifikovaný cíl. Přitom o probíhajícím útoku uživatel napadený malwerem nemusí ani vědět.

Uživatel se stává takzvaným bótem. Tímto se stává botnet síť velice velkou hrozbou. Samotný útočník stojící za útokem a ovládající botnet síť je jen těžko vystopovatelný, poněvadž útok provádí samotní bóti.[1]

Dnes již botnet síť může tvořit jakékoliv zařízení, které je připojené k síti. Hlavním problémem dneška se stává IoT zařízení, které mnohdy nemají žádnou ochranu, nebo jen velmi malou a stávají se z nich bóti, díky kterým rapidně roste síla útoků. Botnety se zvětšují jak ve velikosti, tak v číslech. Botnet síť se stává jednou z největších problematik Internetu celkově. Předpokládá se, že 15-25 všech zařízení, které jsou připojených k Internetu, již tvoří botnet síť. [1]

V okamžiku útoku vyjdou od útočníka instrukce a tisíce, nebo spíš desítky či stovky tisíc počítačů začnou systematický útok.



Obrázek 2: Botnet síť

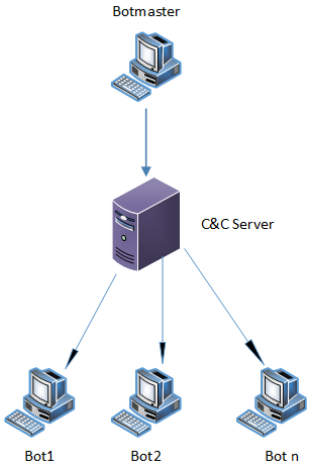
Od roku 1999, kdy byl oficiálně zaznamenán první typ DDoS útoku se dnes již velice zjednodušily podmínky pro zhotovení útoku. Útočníkům dnes již stačí pronajmout si botnet síť. Útoky se účtují podle síly, ale také podle počtu hodin či dnů, cíle a rizika. Možné je také dnes použít volně přístupné aplikace, které generují tyto útoky.

I přes snahu všech počítačových odborníků o snížení počtu útoků a incidentů s tímto útokem spojených, DDoS útoky rostou a to v přímé souvislosti s botnet sítěmi. Důvod nárůstu je lehce dosažitelná a poměrně levná distribuce útoků. Na přímo s tím roste také intenzita těchto útoků.

3.3.1 Typy botnetů

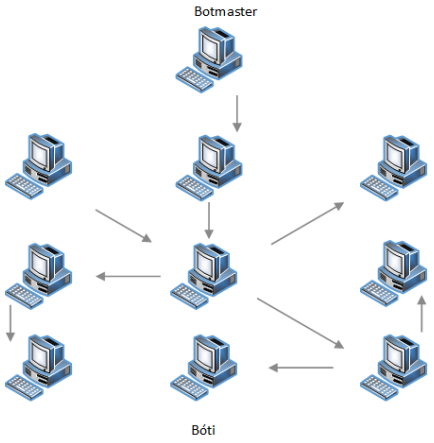
Botnet síť může být centralizována, nebo decentralizovaná. To znamená, že příkazy k provedení dané operace, nebo útoku jsou posílány buď z centralizovaného serveru C&C, nebo jsou bóti

distribuováni z jednoho na druhého. V obou případech se komunikace snaží zůstat skryta a splynout v normálním provozu. [1]



Obrázek 3: Centralizovaná Botnet síť, zdroj:[1]

V decentralizované verzi nemá botmaster plnou kontrolu nad zprávami mezi bóty. Obecně vzato existuje 5 typů botnetů založených na jejich komunikačním kanále a platformě IRC, HTTP, P2P, Cloud a mobilně založené botnety.[1]



Obrázek 4: Decentralizovaná Botnet síť, zdroj:[4]

IRC bóti používají IRC kanály, nebo protokoly ke komunikaci s jejich botmastery. Botmaster může poslat příkaz bótům ve stejném kanále, který používají právě bóti, což je nazýváno push mechanismem.

P2PBotnet nemá hierarchii, ale všichni bóti jsou C&C servery tak, jako bóti. Jakýkoliv z nich může poslat nové instrukce, které jsou pak přeposílány na bóty.

Cloudoví bóti se nacházejí v cloudu, který útočník získal za jiným účelem. Výhody cloudových botů jsou snadné a rychlé nastavení botové sítě virtuálních strojů, všechny jsou k dispozici po celou dobu a mnoho poskytovatelů cloudu nemá způsob, jak detekovat tyto bóty.

Mobilní botnety využívají služby Bluetooth a SMS v chytrých telefonech pro komunikaci. Používají se pro shromažďování dat ze zařízení uživatelů, spíše než pro odesílání nevyžádané pošty nebo pro útoky.[1]

IoT nabírá na popularitě a jsou velmi oblíbené mezi útočníky, protože jsou neustále online, mají výchozí hesla a neobsahují žádný antivirový software.[1]

3.3.2 IP spoofing

S problematikou botnet sítí úzce souvisí problematika podvrhování IP adres, nebo-li někdy označovaný jako IP spoofing. IP spoofing v informatice označuje IP datagram, kde byla falešně vytvořena zdrojová IP adresa. Následně je tento IP datagram poslán k cíli, tím je zatajena identita. IP spoofing slouží často k provedení útoku DDoS.

Internet je založen na IP adresách, ty pracují s IP datagramy. V každém IP datagramu je zdrojová IP adresa a cílová IP adresa. Zdrojová IP adresa má IP adresu odesílatele a tím pádem odesílatele identifikuje. Falšováním, nebo-li spoofováním dosáhneme toho, že IP adresa odesílatele je zamaskována adresou, kterou IP datagramu podvrhneme. Samotný cíl pak obdrží IP paket s podvrhnoutou IP adresou a odpovídá na onu zfalšovanou adresu.

Obrana proti IP spoofingu je relativně snadná a to filtrováním IP datagramů na rozhraní sítě pomocí firewallu. Takovým firewallem obvykle blokujeme IP datagramy, které mají zdrojovou adresu nacházející se uvnitř sítě. Ideální případ je, kdy firewall filtruje také odchozí provoz a je povolen jen takový provoz, který je povolen jen z určitého rozsahu, který je používán. Další možnou obranou a zároveň doporučením je nepoužívat autentizaci jen na základě ověření IP adresy.

Existují dva typy podvrhnutých IP adres:

- Náhodné IP adresy
- Reálné IP adresy

Náhodné adresy jsou adresy, které jsou zcela náhodné a v mnoha případech i neexistující. Oproti tomu Reálné IP adresy jsou adresy, které nemají s útokem nic společného a povětšinou o útoku ani nevědí. Pokud však útočník využije botnet síť k útoku, nemusí vůbec využívat maskování. Na jeho příkaz zaútočí bóti záplavou paketů na oběť pod samotnou kontrolou útočníka.

Počet útočících strojů je různorodý v závislosti na typu oběti. I útoků na síťovou infrastrukturu bývá počet útočníků co možná nejvyšší a proto se z velké části používají botnet sítě.

Podle zroje[10] je znám útok, nazýván Mariposa, který se odehrál v roce 2010, kde podle dostupných informací útočilo přes 15 miliónů unikátních IP adres z celého světa.

4 Typy síťových počítačových DoS/DDoS útoků

Je hodně kritérií, pomocí než můžeme klasifikovat DDoS útoky, je to dáno převážně tím, že máme velkou škálu útoků na různé služby. Jaký útok bude proveden záleží na cíle útočníka. Převážně rozdělujeme DDoS útoky na základě popisu oběti do tří skupin.

Cíl může být:

- Síť
- Server
- Aplikace

Útoky na síťovou infrastrukturu jsou obecně nazývány jako volumetrické útoky a to proto, že používají záplavu paketů. Útočník usiluje o to, aby využil celé dostupné pásmo oběti a znemožnil legitimnímu provozu ke komunikaci s napadeným zařízením.

Rozpoznáváme taky mechanismy vzniku záplavy:

- Přímé záplavy – Paket, který se nezměněn přenáší přes síť až k oběti (UDP nebo ICMP flood)
- Reflektivní útok – Využívají se podvržené IP adresy, útočník tímto skryje svou skutečnou IP adresu, jako útočník pak vypadá nic netušící třetí strana.
- Amplifikační útok – Dovoluje útočníkům vyslat k oběti více paketů, než kolik by sám dokázal vygenerovat. Tak jako v předchozím případě se využívají podvržené IP adresy (NTP a DNS attack)

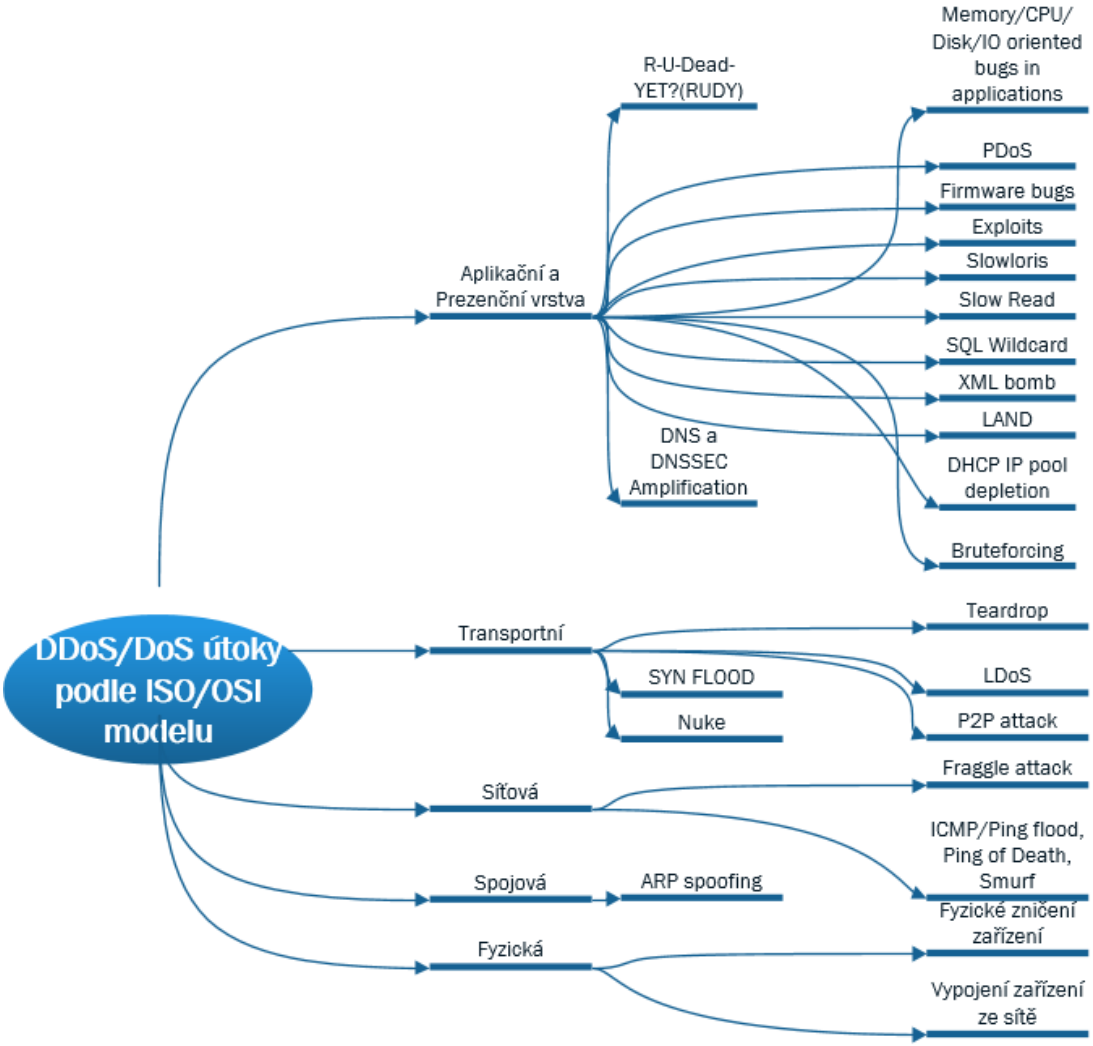
Volumetrické útoky typu DDoS jsou silnější, protože využívají více než jednoho hosta. Většina služeb může díky jednoduchým technikám jednoduše bojovat s některými typy útoku odmítnutí služby.

K nejsilnějším útokům patří kombinace Amplifikačních a Reflektivních útoků. Útočník se doptává pomocí podvržených paketů. Následná reflexe spočívá v tom, že odpověď pak není zaslána útočníkovi, ale oběti, díky podvrhnuté IP adrese.

Útoky na server mají jednoduchý cíl, a tím je využít maximálně procesor, nebo paměť serverů, a tím přímo zapříčinily odmítnutí služby. K tomuto je často použita zranitelnost zařízení. Samozřejmě na tyto útoky nejsou náchylné pouze servery, ale také jakákoliv další zařízení.

Dnes je již známa spousta druhů útoků a některé stále přibývají. Sofistikovanost útočníků takřka nezná mezí, pořád se seznamujeme s novými technikami útoků a chyb, které byly k útokům zneužity.

Na obrázku níže jsem se snažil zachytit všechny známější DoS/DDoS útoky, podle ISO/OSI modelu. Jak ale vidíte, je jich opravdu hodně a důležité je zmínit, že zejména univerzální obrana proti všem těmto útokům neexistuje.



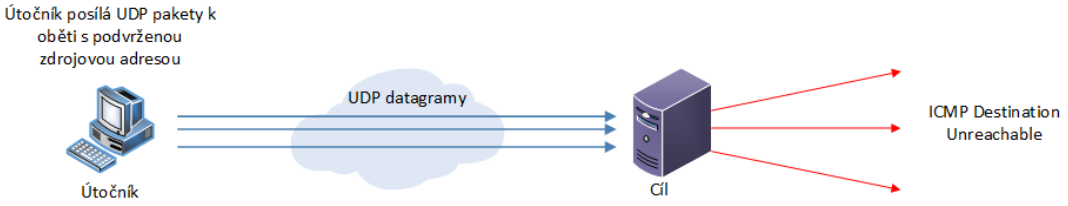
Obrázek 5: Typy DoS/DDoS útoků podle ISO/OSI modelu

4.1 UDP Flood

Pod zkratkou UDP se skrývá tzv. User Datagram Protocol. UDP protokol má na starosti komunikaci v síti, podobně jako protokol TCP, avšak oproti TCP protokol UDP nepoužívá ověřování navázání spojení. Díky tomu je sice rychlejší, ale na druhou stranu také jednoduše zneužitelný.

Útočník může pomocí UDP protokolu provést úspěšný útok, a to tak, že pošle velké množství UDP paketů na náhodné porty cíle. Například reakce, serveru je taková, že na pakety reaguje, nejdříve zkontroluje, jestli porty sleduje nějaká z aplikací. Jestliže tomu tak není, server reaguje. Dává odpověď, že cílová stanice není dostupná. Tuto zprávu posílá zpět paketem ICMP, který je určen pro odesílání chybových zpráv.[4]

Zpravidla by měl mít snahu odpovědět na všechny takovéto dotazy, tedy na všechny UDP pakety, které přijme. Útočníkův záměr je tedy zřejmý. Odesláním obrovského množství UDP paketů na server, ten se bude snažit na všechny odpověď, a vyčerpá tak svou internetovou konektivitu, ale také například zaměstná CPU natolik, že server bude zcela přetížen a jakákoliv další komunikace se stane neproveditelnou.[11]



Obrázek 6: UDP Flood

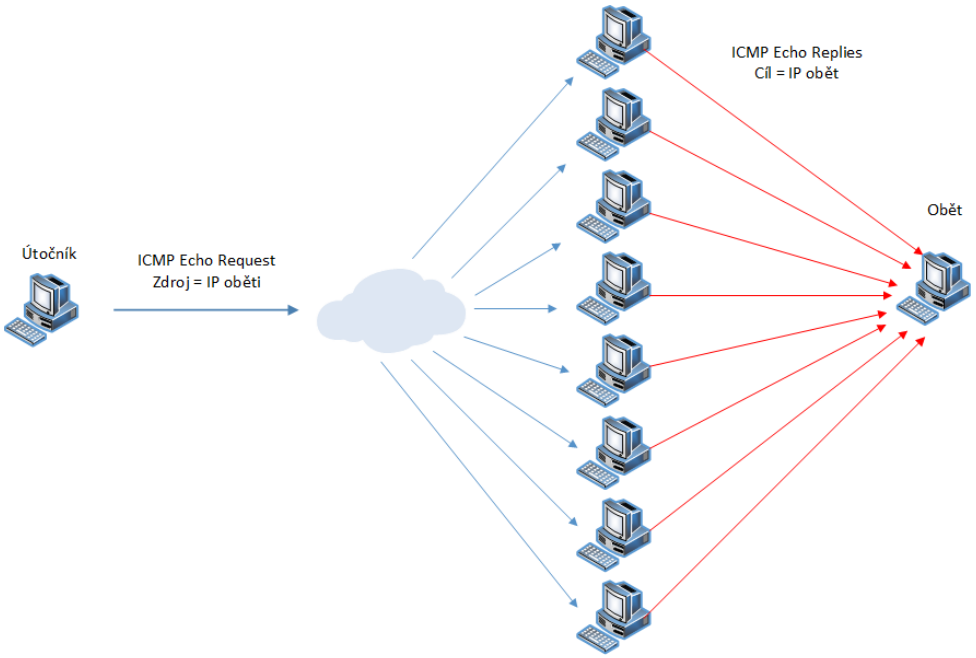
Jeden ze základnějších principů, jak se proti UDP flood bránit je množství ICMP odpovědí. Avšak to může mít neblahý dopad na síť.

Tradičnější metodou je spoléhání na firewally, které odfiltrují nebo přímo blokují škodlivé UDP pakety. Avšak v dnešní době se tento typ obrany stává méně podstatným, neboť moderní útoky s velkým objemem dat mohou tyto firewally překonat. [11] [15]

4.2 ICMP Flood

Jednoduchý druh útoku, který slouží většinou k provedení DDoS útoku, při kterém je oběť zahlcená ICMP Echo Requesty. Protokol ICMP je jedním ze základních internetových protokolů. Je určen k odesílání různých chybových hlášení a pro tento účel ho především používají počítače, servery v síti. ICMP odesílá jednocestné odkazy a přitom není používáno žádné ověřování. Útočník je proto schopen provést DDoS útok.

Nejefektivnější způsob je, kdy útočník nastaví u posílání ICMP paketů možnost záplavy, pakety se poté posílají co nejrychleji, aniž by čekali na případnou odpověď od daného cíle. Útočník si klade za úkol, aby oběť začala odpovídat na ICMP dotazy a zahltila si tak šířku pásma pro odesílaní, ale také příjem dat. Možnost je také, že útočník nemá dost výkonnou CPU jednotku, zatížením se stane nepoužitelná pro další činnost. Normálně je Ping Request používán k testování konektivity dvou či více zařízení.[11]



Obrázek 7: Ping flood

ICMP Flood způsobuje přetížení cílového zdroje pomocí paketů ICMP Echo Request, které se obecně posílají, aniž by čekaly na odpovědi. Protokol posílá neobvykle množství paketů do cílového serveru, ten reaguje a snaží se odpovědět na každou ICMP žádost, a to ve výsledku může vést k odepření služby. Výsledek útoku směřuje ke zpomalení služby a v krajních případech až k odpojení od sítě. Jedná se o běžný záplavový útok.[11]

V dnešní době je poměrně lehké provést ICMP Flood. Provedení útoku také závisí na tom, zda-li útočník zná IP adresu cíle. Útok proto může být rozdělen do tří kategorií založených na tom, zda IP adresa je útočníkovi známa.

1. Cíl na známý cíl
2. Útok na směrovač
3. Blind Ping Flood

Při prvním zmíněném útoku útočník zná IP adresu počítače, na který přímo útočí. Většinou je to jeden počítač v místní síti. Útočník má fyzický přístup k počítači a zjištění IP adresy pro něj tedy není problém. Úspěšný útok poté vede k odstavení počítače.

Druhý zmíněný útok je veden na router, aby narušil komunikaci mezi počítači v síti. Útočník vede tenhle útok většinou proto, že pozná IP adresu lokálního směrovače. Úspěšný útok pak vede k odstavení všech počítačů připojených k routeru.

Blind Ping Flood, tento třetí typ zahrnuje použití externího programu k odhalení IP adresy cílového zařízení. Aby byl útok úspěšný, musí mít útočník k dostupnosti větší šířku pásma, než

oběť. To omezuje schopnost provést útok, zejména tedy proti velké síti. V dnešní době se nabízí provedení útoku za pomocí botnetu, která má tím pádem mnohem větší šanci na úspěch.

Metody obrany jsou různé, mezi ně patří správná konfigurace firewallu, kde zabráníme pingům z vnější vaší sítě, samozřejmě to může mít neblahé následky na síť zahrnující například neschopnost diagnostikovat problémy serveru. Různé světové společnosti provádí ochranu proti DDoS tím způsobem, že omezuje počet požadavků jen na počet, který je povolen. [11]

4.3 TCP SYN Flood

Když jsem výše zmiňoval protokol UDP a jeho zneužití, tak také další z protokolů, protokol TCP, který má na starosti komunikaci v síti, je zneužíván. TCP SYN flood útok tkví v samotné podstatě protokolu nazývaného free-way handshake. V češtině také někdy označované jako třísměrné potřesení rukou. Tento typ útoku má za úkol ověřit, zda obě strany skutečně stojí o spojení. Zároveň je tento typ jeden z nejvíce frekventovaných útoků a současně jeden z nejjednodušších ze všech. Cílem tohoto útoku není získat data, nebo prolomit systém. Usiluje pouze o přechodné funkčnosti určité služby, jako například HTTP.

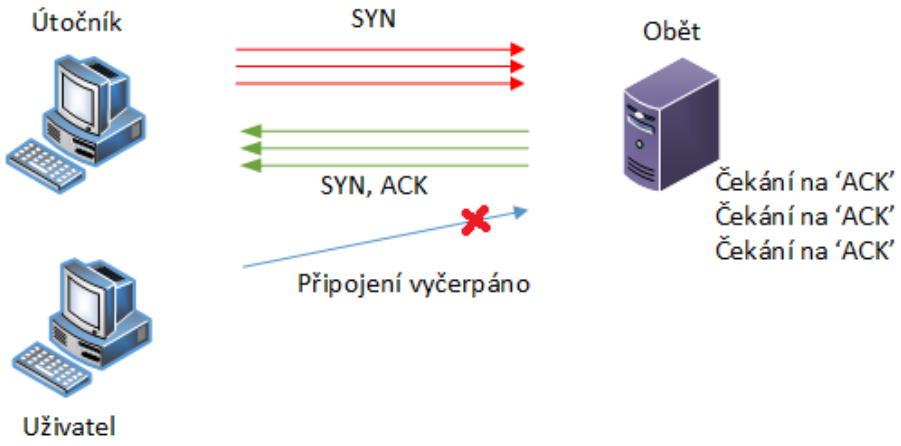
I když je tento útok dobře znám již téměř přes deset let, ukázalo se, jak moc účinný tento typ útoku je a jak nepřipravené firmy, jejichž byznys je mnohdy závislý, na webu jsou. Útok SYN flood byl před pár lety také veden v České republice na banky a byl velice účinný, přitom bychom čekali u bank nejvyšší míru zabezpečení. Proto je mnohdy velice využíván tento typ útoku.

Tento útok spočívá ve snaze o navázání falešného připojení. Napadené zařízení se pro nadcházející připojení snaží vyhradit určité prostředky a tím vyčerpá systémové prostředky zařízení. Napadené zařízení poté může přestat úplně reagovat na jakékoli další pokusy o připojení. [11]

Jak probíhá normální free-way handshake:

1. Klient žádá připojení posláním SYN (synchronizovat) serveru
2. Server odpoví příznakem ACK(acknowledge) a nazpět pak pošle packet s příznaky SYN – ACK
3. Klient v posledním kroku pošle paket ACK

Takto funguje free-way handshake v praxi. SYN flood je velice účinný. Útočník má dva způsoby. První je jednoduchý. Může ACK paket přejít, nebo může poslat paket s falešnou IP adresou a server poté pošle příznaky SYN a ACK úplně jinam, paketu ACK se již nedočká.



Obrázek 8: SYN Flood

Z obrázku můžeme vidět, že předtím, než server ukončí spojení, další paket SYN dorazí na server. To pak zanechává velké množství napůl otevřených spojení a proto také někdy útoky typu SYN flood jsou někdy označovány jako napůl otevřené. Když nakonec tabulka serveru je zaplněná a pokračuje navazování připojení, služba klientů znemožní. V dnešní době jsou servery stále lépe vybaveny a je stále obtížnější udělat zcela úspěšný útok.

Je mnoho technik, jak se bránit proti SYN Flood útoku:

- SYN cookies – použitím kryptografického hashování server pošle svou odpověď SYN-ACK se sekvenčním číslem, které je tvořeno z adresy IP klienta, čísla portu a případně dalších unikátních identifikačních informací. Toto je součástí paketu ACK, jestliže klient reaguje, server ověří protokol ACK a až potom přidělí paměť pro připojení.
- RST cookies – po prvním požadavku klienta, server úmyslně pošle neplatný příznak SYN-ACK. Což by mělo vést k tomu, že klient generuje balíček RST, který informuje server o tom, že je něco špatně. Pokud je toto přijato, server ví, že je požadavek legitimní a přijme od klienta příchozí připojení.
- Stack tweaking – administrátoři omezí časový limit a tím přímo zmírní SYN Flood, nebo přímo selektivně zruší jednotlivé připojení.

Výše zmíněné prostředky samozřejmě vycházejí z toho, že cílové sítě umí zpracovat obrovské množství dat.

4.4 Slowloris

Tento druh útoku patří mezi další z řady útoků, který je velice zákeřný, těžce detekovatelný, je vynalezen Robertem Hansenem. Útoku stačí i minimální síťový provoz k tomu, aby byl útok

úspěšný. Dokonce jediný počítač je schopný vyřadit službu jako je HTTP běžící na jiném počítači a je možného použít pro efektivní DoS útok.

Program, sestavený v programovém jazyce Perl funguje tak, že uvolní velké množství síťových spojení a jeho cílem je udržet tyto spojení co možná nejdéle. HTTP požadavky serveru se posílají značně pomalu, po částech a blízko limitu vypršení. Server drží tato spojení stále otevřená, samozřejmě ale může držet jen omezený počet připojení, a tak další požadavky jsou odmítnuty. Typ útoku Slowloris je velice účinný proti mnoha populárním webovým serverům, včetně Apache 1.x a 2.x.

Vyjma toho, že tento útok zahltí server, je taky těžko detekovatelný. K žádnému velkému problému nedochází, logy webu proto nic nehlásí. Jediná věc, co by mohla na útok upozornit je větší počet otevřených spojení se serverem. [11] [15]

4.5 HTTP Flood

Záplavové útoky, mezi které patří i HTTP Flood, se snaží vyslat k oběti velké množství paketů, čímž dosahují zahlcení sítě. Útočník při útoku využívá legitimních požadavků HTTP GET nebo POST k útokům na webový server nebo aplikaci. Jedná se o sofistikovaný útok na 7 vrstvu ISO/OSI modelu. Požadují hlubší pochopení útočníkova cíle nebo aplikaci, která má být cílem. Každý útok je specificky tvořen tak, aby byl co nejvíce účinný. To činní z HTTP Flood útoku těžce detekovatelného a hůře blokovaného. [11]

Útok probíhá následovně. Když HTTP klient, jako například webový prohlížeč, komunikuje s aplikací nebo serverem, posílá HTTP Request, obecně jeden ze dvou typů GET nebo POST. GET se používá k zobrazení standardního statického obsahu, jako jsou obrázky, zatímco POST Request se používá pro přístup k dynamicky generovaným zdrojům. Samotný útok je pak ještě účinnější, je-li server nebo aplikace nucena vyhradit maximum ze svých zdrojů pro každou jednotlivou žádost. Útočník se tak zpravidla bude snažit zahltit server s více požadavky, které budou mít velké nároky na zpracování.

Z tohoto hlediska jsou pak tyto útoky pomocí požadavku POST nejvíce účinné, protože požadavky POST mohou obsahovat parametry spouštějící komplexní zpracování na straně serveru. Na druhé straně jsou útoky založené na protokolu HTTP GET jednodušeji vytvářeny a mohou být efektivnější ve scénáři botnetu.[11]

Obrana proti útoku je obtížná zejména proto, že je velice těžké rozlišit od normálního provozu, protože se jedná o standardní URL požadavky. To z nich dělá jednu z největších hrozeb pro bezpečnost web serverů a aplikací. Tradiční detekční metody založené na detekci zjišťování HTTP Flood jsou v tomto případě neefektivní, protože objem provozu na HTTP serverech jsou často na hraně detekčního prahu. Jedním z nejúčinnějšího mechanizmu pak je kombinace metod profilování provozu, identifikace IP a abnormální sledování činnosti.[11] [4]

Zranitelnosti v HTTP protokolu používá také program Rudy. Celý název je pak R-U-Dead-YET a jedná se o útok, který se děje „low and slow“. Rudy útok vytváří menší množství dat, a proto je možné, že proklouzne detekčním prahem většího množství ochranných prvků. Rudy

útočí na formuláře HTTP POST, kterým zasílá neobvykle zdlouhavým způsobem po 1 byte data. To přiměje server, aby držel spojení otevřené. Výsledkem je pak nedostupnost aplikace či serveru.[11]

4.6 Ping of Death

V češtině nazýván jako ping smrti, je jedna z dalších variant útoku. Ping of Death využívá chyby v implementaci protokolu TCP/IP. Jedná se o starší typ útoku, který se již dnes tolik nepoužívá.

Ping se běžně používá pro verifikaci, zda dané zařízení reaguje. Poněvadž hodně počítačů v minulosti nebylo schopno zpracovat ping, který byl větší než 65 535 byte a tato hodnota odpovídá maximální velikosti paketu protokolu IP včetně její hlavičky, byla tato chyba jednoduše zneužitelná útočníkem. Dříve mnoho počítačů jednoduše nemohly přijmout takovýto paket. Útočníkův cíl je zřejmý a velice snadný. Posláním abnormálně velkého paketu do určité sítě, může způsobit chyby. Zpravidla dochází k přetečení zásobníku, které směřuje k selhání systému, a které nejsou imunní proti tomuto útoku. Většinou tyto útoky směřovaly k zamrznutí zařízení. Dříve byly tyto útoky populárnější zejména proto, že útočníkova identita byla lehce zfalšovatelná a také proto, že útočník nemusel mít tak podrobnou znalost zařízení, na které útočí. Stačila mu IP adresa. Záporná stránka útoku je, že dané zařízení většinou jen zpomalíte. Tento druh útoku lze brát jako předka útoku Ping Flood, který jsme si popsali již výše.[11] [15]

4.7 Smurf attack

V informatice označujeme typ DDoS útoku, ve kterém se oběti posílá velké množství ICMP paketů z falešných IP adres. Většina i dnešních systémů má má nastavenou odpověď na zdrojovou IP adresu. To však při obrovském množství dotazů může vést k momentu, kdy počítač je zaplaven natolik, že je neschopen jakékoliv další odezvy. Samotný název smurf pak pochází ze souboru smurf.c, zdrojového útočného kódu, který byl prvně spuštěn roku 1997.[4]

Útok probíhá zasláním ICMP ECHO REQUEST paket na broadcast adresu sítě, adresa odesílatele zfalšovaná na stroj v síti. Na broadcast ping odpoví všechny stroje v síti (100 stanic, 1 paket = zesilovací faktor 100).

S obranou proti tomuto typu útoku pak přišla firma Microsoft a ve Windows NT byla ve výchozím nastavení zablokovaná služba Ping. Zmírnění pak probíhá pomocí dvou způsobů. První, že nakonfigurujeme hosty a směrovače tak, aby správně reagovaly na ICMP žádosti. Druhý, nakonfigurovat směrovače, aby nepřeposílaly pakety přímo na vysílací adrese. Dalším možným řešením je pak filtrace průniku, kdy jsou odmítány pakety na základě podvržených IP adres.[11]

5 Nástroje a Motivace pro útoky, Aktuální situace

5.1 Nástroje pro útoky

Při krátkém hledání lze dnes najít mnoho nástrojů pro generování DDoS/DoS útoku. I méně zkušenému uživateli stačí v mnoha případech zadat pouze IP adresu, zvolit typ útoku a spustit útok. Nástroje LOIC, HOIC a další umožňují plnohodnotný DDoS/DoS útok bez hlubších znalostí protokolů, které využívá.

V této kapitole se zaměřením na pár nejznámějších nástrojů umožňující DDoS/DoS útok.

5.1.1 LOIC

Celým názvem Low Orbit Ion Cannon, název vychází z počátečních písmen. Jedná se o jednoduchý nástroj používán útočníky a hackerskými organizacemi po celém světě. Dokáže generovat velké DDoS/DoS útoky, které spotřebovávají velké množství šířky pásma, a to generováním velkého množství TCP, UDP a HTTP provozu. Tento nástroj byl původně vynalezen firmou Praetox Technologies, avšak tato společnost tento program zveřejnila se zdrojovými kódy. Toho později využila hackerská skupina Anonymous, kódy využila, doplnila a začala používat ke koordinovaným DDoS útokům.[1]

Za krátkou dobu skupina označila tento nástroj za svůj oblíbený DoS nástroj a dokázali ho modifikovat tak, aby bylo možné spouštět útok najednou od všech uživatelů, kteří mají přihlášení odběr IRC kanálu ovládaného Anonymous. Díky IRC byl pak administrátorovi dána do ruky schopnost předávat příkazy LOIC programům běžících na strojích, které jsou přihlášeny k odběru. Takto koordinovaný útok je pak mnohem silnější, než útok, který by závisel na samotných lidech, kteří by útok spouštěli ručně. Avšak nedostatek anonymity v době útoku přiměl Anonymous k ustoupení od téhle původně oblíbeného programu. A od roku 2011 již odstoupili od používání programu.[1]

5.1.2 HOIC

Celým názvem High Orbit Ion Cannon, je stejně jako výše zmínění LOIC jednoduchý nástroj umožnující DoS útok zaměřující se na spotřebování velkého množství šířky pásma. Tentokrát ale umožňuje generovat velké množství HTTP GET a HTTP POST požadavků. Efektivita programu je umocněna použitím skriptů. Ačkoliv ani tento program neposkytuje absolutní anonymitu, lze pomocí těchto zesilovacích skriptů definovat seznam cílů a informací, které bude program procházet a za jejichž pomocí může generovat provoz. Díky této technice se může útočník stát anonymním, protože program HOIC nebude poskytovat reálné informace o uživateli a následná mitigace se stane hůře proveditelnou. Nástroj HOIC je dodnes používán Anonymous při útocích DDoS.[1]

5.1.3 GoldenEye

GoldenEye je testovací DDoS/DoS nástroj využívající aplikační vrstvu. Využívá možnosti KeepAlive (a Connection: keep-alive) spárované s možnostmi Cache-Control, aby přetrvávaly propojení soketů pomocí mezipaměti, dokud je to možné a nejsou vyčerpány všechny dostupné sokety na serveru HTTP/S.

Tento projekt byl zahájen vlivem Barryho Shteimana, který vytvořil HULK, nástroj důkazu koncepce, který se rozhodl zlepšit a který se později stal GoldenEye.

Tento software je napsán čistě v programovacím jazyce Python.[18]

5.1.4 Hping3

Tento program byl vynalezen developerem Salvatore Sanfilippo v roce 2005. Hping3 patří mezi dostupné paketové generátory pro analyzování provozu. Je především určen pro testování, zabezpečení počítačových sítí a ověření připravenosti sítě, routerů, firewallů a dalších na daný typ útoku. Podporuje všechny druhy protokolů.

Hping3 je schopný posílat statisíce paketů za sekundu, jak si ukážeme v praktické části, při kterých můžeme definovat mnoho proměnných, jako například source port, destination port, množství paketů, velikost paketů a mnoho dalšího.

Program je volitelně šiřitelný pomocí GNU GPL, a je určen primárně pro systémy Linux a Solaris.[17]

5.1.5 Ostinato

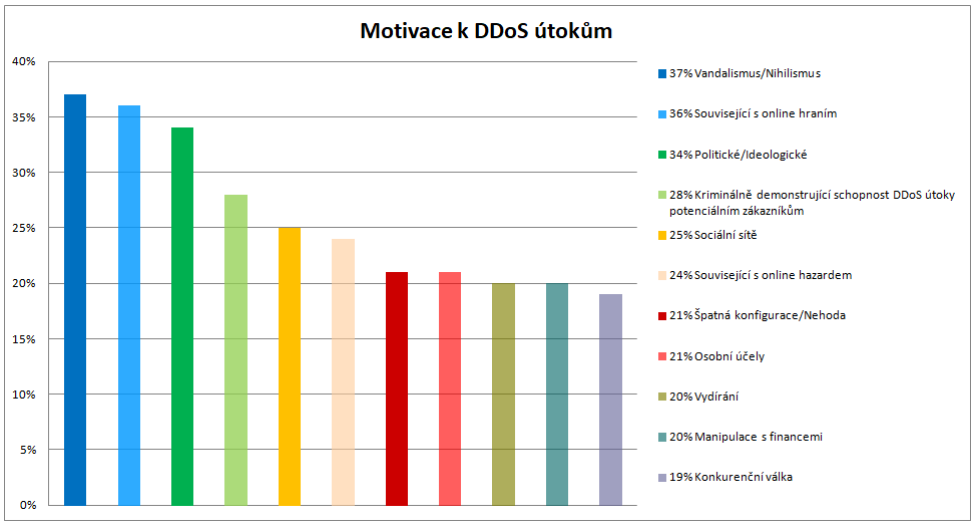
Ostinato je paketový generátor a analyzátor s přívětivým GUI. Slouží také jako Python API pro automatizaci síťových testů a je schopen posílat několik paketových toků s odlišnými protokoly. Slouží k penetračnímu testování.

Ostinato je označován jako cross-platform, je tedy spustitelný na všech platformách Windows, Linux, BSD nebo Mac OS X, což je jedna z jeho největších výhod a také dokáže pracovat s IPv6 na rozdíl od dříve zmíněných paketových generátorů, jako například Hping3, který generuje pouze IPv4 pakety. Ostinato je open-source program. [16]

5.2 Motivace DDoS útoků

Typ útoku DDoS se pomalu ale jistě stává převládajícím typem hrozeb útoků na počítačovou síť. Počet útoků v posledních letech rapidně vzrostl jak v počtu, tak také v objemu. Trend DDoS útoků vede ke kratším útokům, ale s daleko větším objemem dat, než tomu bylo v minulosti.

Útočníci jsou primárně motivováni:



Obrázek 9: Motivace útočníků, zdroj:[19]

- Ideologicky a politicky - také nazýváme tzv. hacktivisty, používají útok na konkrétní webové stránky s kterými ideologicky nesouhlasí. Může to být i například s politickým názorem, nebo ideologií, která určitá strana prosazuje.[19]
- Obchodně – podniky mohou používat útok jako záminku k poškození konkurence, strategicky odstranili například webovou službu konkurentů a sami z toho těžili. Motivace je tedy za účelem zisku, a nebo jako možnost pro odstranění konkurence. Často se stává, že například v době svátků si někdo tento útok objedná, a vyřadí tak jednu z konkurence z provozu. Útok může trvat i několik dnů a většina eshopů na útok není dostatečně připravena. Daný útok je pro ně pak osudný a eshop poté přichází o značný výdělek.[19]
- Vandalismus - také nazývaní „skript-kiddies“ používají předepsané skripty, nebo předem určené programy o kterých nemají hlubší znalost, jak fungují k zahájení útoku. Pachatelé útoku obvykle zkoušejí, co dokážou, respektive nástroje, které zkouší k realizaci útoku.[19]
- Vydíráním - pachatelé používají útoky, nebo hrozí útoky jako prostředek k vydírání peněz svých případných obětí.
- Kyber terorismus - Tyto útoky jsou vedeny útočníky, kteří vedou svůj útok hlavně na systémy, které jsou kritické pro fungování státu, kritickou infrastrukturu.
- Skrytí sekundárního útoku - Projevení nesouhlasu a skrytí sekundárního útoku jsou nejčastější z motivů, často se vede útok na klíčovou infrastrukturu, která trvá i dny, aby zamaskoval jiný probíhající útok. Ví-li útočník o chybě v systému, kterou by při využití mohl vzbudit zájem u správců sítí, využije k maskování DDoS útok. Útočník pak má dost času na využití chyby a správcům většinou trvá delší doba, než odhalí průnik do systému. Při úspěšném útoku může dojít k nucenému restartu stroje, po kterém na stroj byl nainstalován jiný škodlivý software, které se právě po restartu projeví.

Podle expertů v posledním desetiletí začíná být největší podíl útoků politický hacktivismus. DDoS se také v nacházejících letech stává nástrojem vydírání. Většinou oběť bývá kontaktována mailem a je mu vyhrožována, jestli nezaplatí určitou částku, tak na něj bude podniknut masivní DDoS útok.

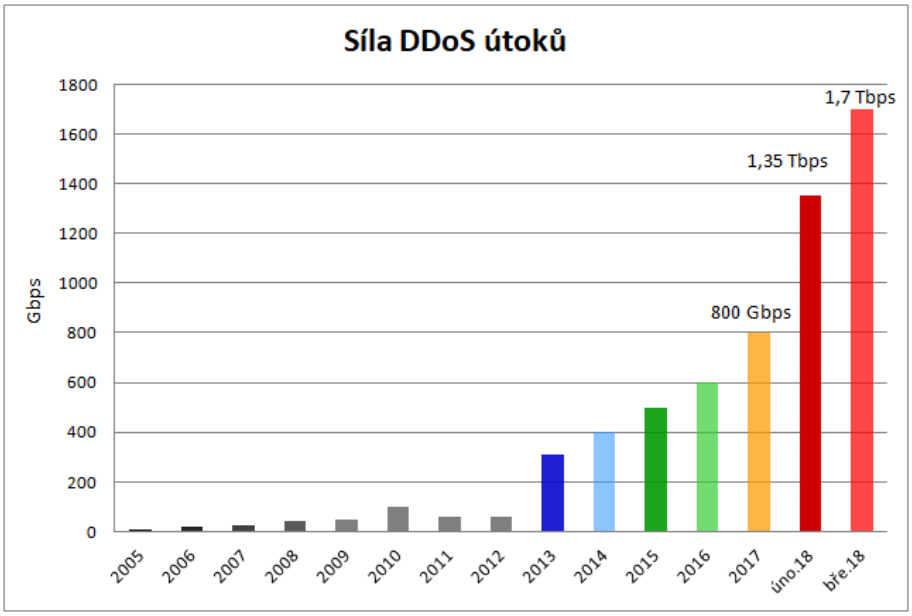
5.2.1 Útoky na objednávku

Útoky na odmítnutí služby se stávají čím dál populárnější taky díky jeho jednoduchosti a také díky cenové dostupnosti. Toto potrvzuje i autor článku[5]. V dřívějších dobách se sítě typu botnet používaly jen pro vlastní potřebu, zábavu a velikosti těchto botnet sítí pak udávala jen útočníkovu prestiž v komunitě hackerů. Postupem času, a hlavně díky možné zneužitelnosti sítě typu botnet, se z toho stal slušný byznys. Se zvyšující se popularitou se na diskuzních fórech začaly objevovat nabídky pronájmu těchto botnetů. V praxi to pak vypadá velice jednoduše. Zájemce o útok jen zadá cíl a pošle peníze na účet a o všechno ostatní se stará útočník, který službu nabízí.

V dnešní době stačí pouze zadat slovní spojení jako je například *DDoS for hire* nebo *DDoSer* a najdeme hned několik nabídek na DDoS útok. Cena útoku je přímo úměrná velikosti botnetu, doby útoku a taky cíle. Samozřejmě cena se odvíjí od různých proměnných, například je-li cíl nějaký na první pohled nepropracovaný a nezabezpečený e-shop, nebo je-li cílem nějaký server pod státní správou, v ten moment útočník samozřejmě víc riskuje a je potřeba si připlatit a to někdy značně. Útok na jednoduché cíle pak stojí opravdu pár dolarů. Na některých stránkách si lze na některé cíle objednat DDoS útok zcela zdarma.

5.3 Aktuální situace

Od roku 2005 počet útoku stále narůstá a do roku 2010 největší útoky dosáhly maximálně 100 Gbps.[19]



Obrázek 10: Síla DDoS útoků od roku 2005 do roku 2018

V roce 2013 největší útok vzrostl na 309 Gbps a nadcházejícím roce to bylo již 400 Gbps. Nadcházející rok to bylo o dalších 100 Gbps více. V roce 2016 se začalo spekulovat o tom, že útoky dosáhnou okolo 600 Gbps a více.

Podle hlášení Arbor Networks je více než 74% útoků pořád pod hranicí 500 Gbps. Přesto je tato síla útoku alarmující, zejména pro sítové infrastruktury, které nejsou dostatečně zabezpečeny a připraveny na možné DDoS útoky.

Tento rok jsme se poprvé přehoupli přes 1 Tbps, což už je opravdu velká vyprodukovaná síla. Na začátku tohoto roku v únoru firma Akamai potvrdila 1.3 Tbps DDoS útok proti GitHubu a učinil tak nový rekord, podle zdroje[7] na oběť mířilo 126,9 miliónů paketů za sekundu. GitGub reagoval přesměrováním veškerého provozu na služby firmy Akamai, která zajišťuje obranu proti DDoS útokům. Po první vlně pak přišla ještě druhá, která měla už jen 400 Gbps. Za rekordem stojí vzestup DRDoS, kdy útoky nabývají na velké síle. Avšak ani ne měsíc poté v březnu tohoto roku, byl zaznamenán ještě silnější útok. Cíl útoku byl americký Internetový poskytovatel. A byla použita opět reflektivní metoda útoku pomocí otevřených memcached serverů, které mohou útok znásobit až padesátitisíckrát.

Společnost prý dokázala útok ustát, hlavně díky předem připraveným procesům k řešení takových útoků. K čemuž se dostáváme k tomu, že nejlepší obranou před DDoS útoky je samotná prevence.[8]

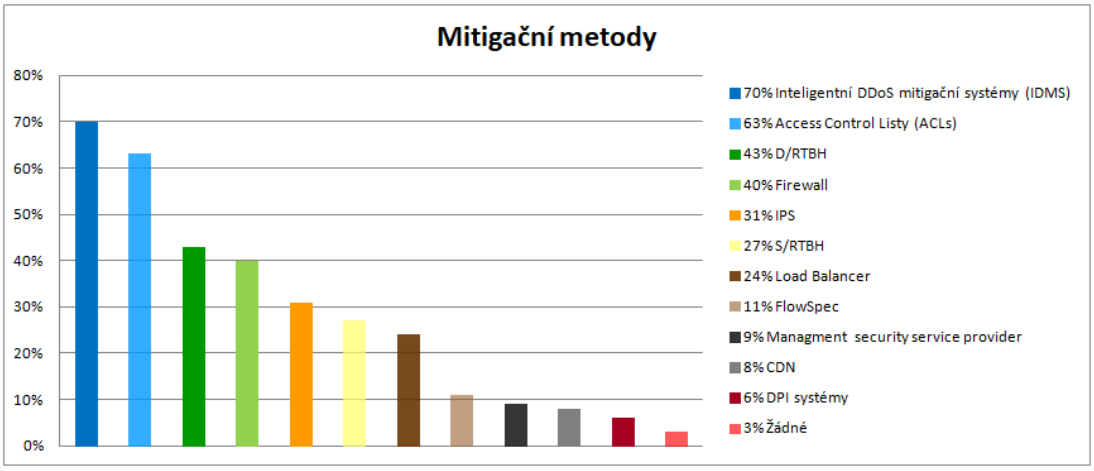
Jeden průměrný útok trvá méně než 30 minut. Průměrná ztráta pro internetové obchody a eshopy je většinou tak vysoká, že většině společnosti se dnes většinou vyplatí investovat do prevence útoku.

6 Metody obrany před síťovými DoS/DDoS útoky

Účinná obrana proti DDoS útokům obsahuje zejména prevenci. Ta je velice důležitá a vede ke snížení rizik a také k nárůstu bezpečnosti samotného systému. Mezi ně bezesporu patří Ingress Filtering, zajištění bezpečnosti koncovým zařízením a hlavně instalace záplat pro systém. Pro bezpečnost systému je včasná instalace záplat nezbytná, neboť systém může být napaden skrz chybu, která ještě není známá a i dobrá prevence a systém na odhalení útoku nemusí být dostatečně efektivní, aby případnému útoku odolal. [1] [19]

Nedílnou součástí systému je také včasná detekce a reakce na útok. Pro detekci útoků se používají různé nástroje a analyzátory, jako jsou například NetFlow, nebo logy ve firewallu, IPS/IDS a SIEM.

Mezi reakční systémy můžeme zařadit Access Control listy (ACL), Firewally a IPS systémy.



Obrázek 11: Mitigace DDoS útoků, zdroj:[19]

6.1 IDMS

Celým názvem Intelligent DDoS Mitigation Systems, je nejpoužívanější mitigací DDoS útoků. Jedná se o velice pokročilé IPS zařízení, které se specializuje přímo na tento druh útoků. Průběžně podrobně analyzuje síťový podezřelý provoz, který je na něj přesměrován. Tento typ ochrany efektivně zastavuje jak volumetrické, tak aplikační útoky. IDMS musí být bezestavový. Většinou jsou nasazena u ISP, na hranici sítě nebo datového centra.[6]

6.2 Firewall

Doporučuje se používat firewall. Útočníci vytrvale hledají díry v systémech, zda se v nich nevyskytují známá zranitelná místa. Síťové firewally (ať už softwarové, nebo hardwarové) poskytují určitou míru ochrany před těmito útoky. Žádný firewall však nemůže odhalit nebo zastavit všechny útoky, takže nestačí jen nainstalovat firewall a ignorovat všechna ostatní bezpečnostní

opatření, ale taky vhodně nastavit bránu firewall a blokovat veškerý nežádoucí provoz a minimalizovat tímto riziko průniku útočníka do sítě. Vhodným a včasným nastavením či přenastavením pravidel firewallu je možné zamezit většině útoku. K detekci se často používá IDS.

6.3 ACL

Access Control List, častěji se můžeme setkat jen se zkratkou ACL, je určitý, specifikovaný seznam pravidel, která řídí přístup k nějakému objektu. ACL jsou velice často používány v řadě aplikací, často také u aktivních síťových prvků, ale třeba i u operačních systémů při řízení přístupu k objektu. Pokud někdo požaduje přístup k nějakému objektu, tak se nejprve zkontroluje ACL přiřazený k tomuto objektu, zda je tato operace povolena a popřípadě komu.

ACL listy můžeme použít kupříkladu na koncových routerech. Tato bezestavová pravidla je možné využít v případě, že není saturovaná linka ven. Pokud je DDoS útok možné snadno filtrovat třeba podle IP adres nebo cílového portu. Je tak možné tok na konkrétní server zastavit například v datacentru.

6.4 IDS

Firewally nedokážou dostatečně chránit a tak s vyvíjejícím se trendem informačních sítí, a tím přímou úměrou rostoucí sofistikovaností útoků, bylo nutné zavést systém IDS. Protože ani ty nejvýkonnější a nejlepší nebyly schopny včas a dostatečně reagovat na probíhající útok. Systém IDS monitoruje a současně upozorňuje v real-time čase na potenciální nebezpečí, které jsou vyhodnocovány podle předem definovaných pravidel. Právě proto, že nezasahuje aktivně do komunikace, je tento systém často nazýván jako pasivní. V praxi se často můžeme setkat s případem, kdy systém IDS je spojen s firewallem. Když systém vyhodnotí potenciální nebezpečí, zašle se dotaz na firewall a je generováno pravidlo.[1]

6.4.1 NINDS

Je nejvíce rozšířeným druhem. Tento systém je připojen do určité části sítě, naslouchá veškerému provozu. Data, která NIDS získá, posílá na server, kde se uchovávají například v relační databázi. Senzory bývají umístěné na zařízeních, jako například Switch, které jsou konfigurované k zrcadlení provozu. Systém zkoumá všechny pakety a snaží se objevit škodlivý provoz.[1]

Zpravidla jsou na předem nadefinovaných serverech, zpracovávají informace přijaté ze síťových rozhraní. Podstatné je jejich umístění, aby zachycovaly co největší, ideálně všechen, síťový provoz. Takovýto program je například Snort, který se velmi často používá s programem Suricata.

6.4.2 HINDS

Je dalším druhem IDS, je to tzv. Host-based Intrusion Detection Systém, který se chová podobně jako NIDS. Většinou se nachází na síťových zařízením na kterých je chtěno mít detailní infor-

mace o anomáliích. IDS orientován na hostitelský systém. Shromažduje informace z nějakého konkrétního systému. Takový je například na Linuxu LIDS.[1]

LINDS je softwarová záplata Linuxového jádra, která umožňuje významně zvýšit zabezpečení Linuxu.

Obecně má dva primární úkoly:

- Podporu při včasném objevení napadení systému a omezení práv superuživatele (root) tak, aby nemohl způsobit v systému velké škody.
- Tím se sice administrátor připravuje o část svých práv, ale zároveň jsou tato práva odepřena i útočníkovi.

6.5 IPS

Tento systém je brán jako rozšíření systému IDS. Systém dokáže nejen detekovat neobvyklé aktivity, ale taky dokáže proti nim aktivně vystupovat, proto je tento systém nazýván aktivním. Dokáže okamžitě reagovat na případné hrozby a zamezit jim. V porovnání s IDS, IPS provádí případné úkony zcela automaticky. Problém může nastat v moment, kdy se jedná o tzv. false positive, tedy o planý poplach. Tam, kde by mohl správce vyhodnotit hrozbu jako neškodnou, IPS zareaguje stejně jako proti případnému útočníkovi. V rámci IPS musí být konfigurace velice citlivá a přizpůsobená potřebám každé sítě, aby poplachů bylo co možná nejméně, ale bezpečnost sítě byla co možná nejvyšší.

IPS se na rozdíl od IDS nasazuje in-line, podobně jako firewall. Díky tomu by přes něj měl projít veškerý provoz. Není závislé na jiných zařízeních a útok může blokovat okamžitě, nikoli až poté, co část paketů již pronikla do sítě.

IPS může provádět takové akce jako vyvolání poplachu, filtrování škodlivých paketů, násilné resetování spojení a/nebo blokování provozu z podezřelé IP adresy. Všechny tyto úkony často provádí ve spolupráci s firewallem. IPS také umí opravit chybný cyklický redundantní součet (CRC), defragmentovat proudy paketů, předcházet problémům s řazením TCP paketů, a čistit nežádoucí přenos včetně nastavení síťové vrstvy.

6.6 Honeypot

Jedná se o zajímavé rozšíření, které lze aplikovat na IDS/IPS systém. Jedná se o virtuální systém se svou IP adresou, který se jeví útočníkovi jako lehká kořist. Při napadení systému je útočník okamžitě odhalen.

Honeypoty jsou taky využívány k detekci malwaru. Malwary pořád mění své chování a toto chování je potřeba analyzovat. Z toho důvodu je nutné tyto malwary nějak nalákat. Získané informace mohou posloužit například pro aktualizování antivirových programů. Honeypoty se někdy združují do sítí, tzv. Honeynetů.

7 Simulace útoku bez ochrany koncového zařízení

Úkolem praktické části diplomové práce je návrh systémů pro detekci a potlačení útoků využívajících odepření síťových služeb a následné ověření tohoto systému v laboratorním prostředí. Prvně si ukážeme možnosti laboratorního prostředí, prvků a aplikací, které k útoku budeme využívat, a jejich simulací na náš cíl, který bude Apache2 Server bez jakékoliv ochrany.

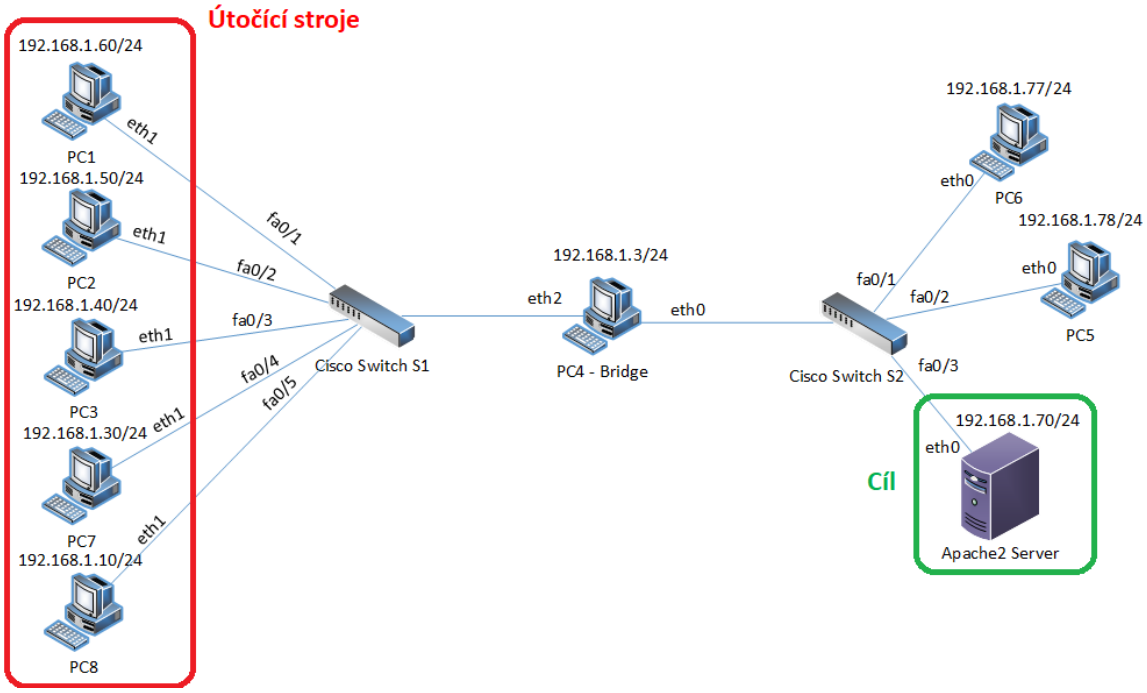
V práci bude taktéž zaznamenán popis útoku a záznam jak v paketovém analyzátoru Wireshark, tak také zobrazení v programu nload, který slouží jako konzolová aplikace, která sleduje provoz sítě, využití šířky pásma v reálném čase a vizualizuje vstupní a výstupní provoz.

Praktická část práce byla rozdělena na dvě části. V první části bude simulováno DDoS útok bez jakékoliv ochrany koncového přístroje Apache2 Server. Tento průběh bude zaznamenán a popsán.

Druhá část bude zaměřena na Anti DDoS open-source systém, jeho nasazení a ověření v laboratoři.

V sekci Nástroje pro útoky bylo zmíněno mnoho programů pro uskutečnění úspěšného DoS/DDoS útoku a v práci byl použit jeden ze zmíněných programů, program hping3, který po krátkém nainstalování na OS Ubuntu 14.04 plnohodnotně vykonává svoji funkci.

7.1 Topologie



Obrázek 12: Topologie pro simulaci útoku bez ochrany koncového zařízení

Takto vypadalo moje schéma zapojení pro praktickou část práce. Na levé straně byly připojeny počítače PC1-PC3,PC7,PC8 do switche - S1. Oba použité switche S1 a S2, byly značky Cisco s označením Catalyst 2950. Z počítačů simulujeme útok na Apache2 Server. PC4 nám souží jen jako prostředník mezi S1, S2 a Apache2 Serverem, a byl přidám pro následné ověření funkčnosti našeho programu. Z PC4 byl udělán bridge, který přemosťoval komunikaci mezi S1 a S2.

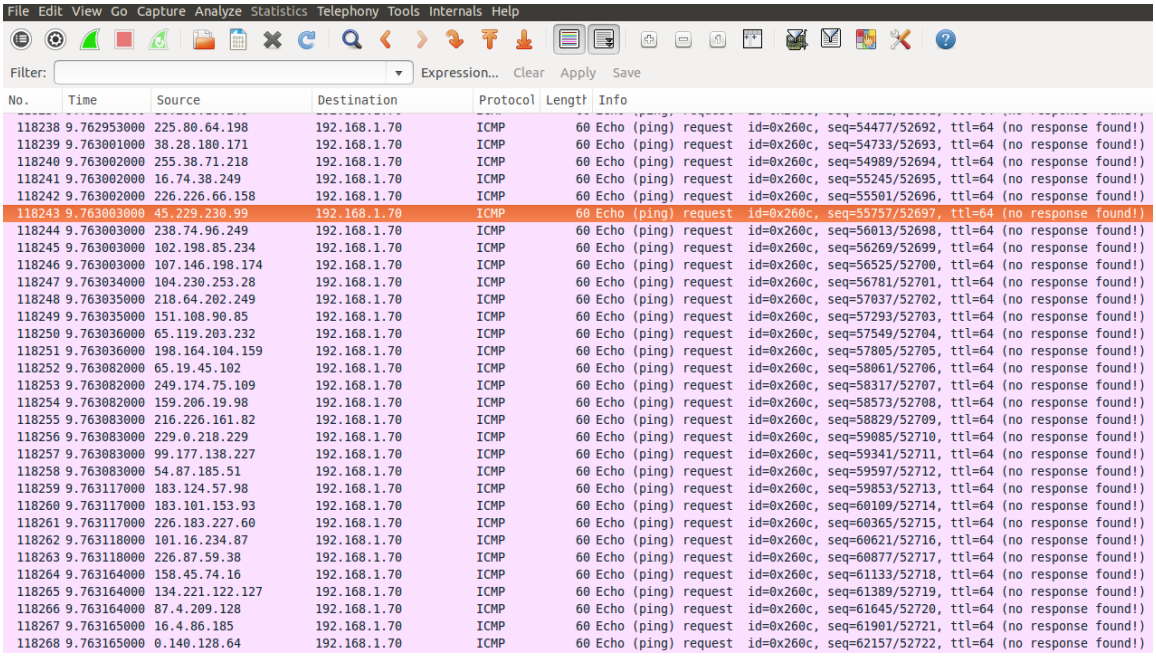
7.2 ICMP flood

Pro útok jsem na všech útočících zařízeních použil příkaz:

```
hping3 --flood -1 --rand-source 192.168.1.70
```

Pro útok bylo použito 5 počítačů se schématem topologie viz. obrázek [12].

Příznak *–rand-source* nám pomůže v realističtější simulaci DDoS útoku, neboť generuje obrovské množství náhodných podvržených IP adres, které simulují útok. Výpis z Wiresharku s příznakem *–rand-source* na napadeném stroji pak vypadal takto:

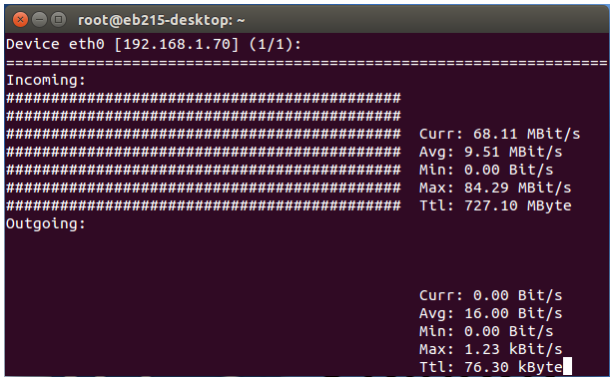


Obrázek 13: ICMP flood s podvrženými adresami - výpis z Wiresharku

Ze sloupce Source je zřejmé, že se nám pomocí položky v příkazu *—rand-source* podařilo úspěšně podvrhnout IP adresy útočníka a zaplavili Apache2 Server ICMP žádostmi ze statisíců podvrhnutých adres v rámci pár sekund. Sloupec Destination nám jen potvrzuje, že cílem byl Apache2 Server s IP adresou *192.168.1.70*. Tímto se nám zcela zahltila šířka pásma a nebylo možné s počítačem úspěšně navázat spojení. Spuštěním jakéhokoli paketového analyzátoru jsme zcela zaměstnali práci všech čtyř jader procesoru, již po pár sekundách nebylo možné s Apache2 Serverem vůbec pracovat. A byl vynucen restart. Avšak z výpisu je zřejmé, že v rámci našeho

testu Apache2 Server na útok nereagoval a neposílal zpět odpovědi na žádosti, což se nám jen potvrzuje ze sloupce Info *no response found* ani u jedné z adres dotazujících se pomocí protokolu ICMP na dostupnost počítače.

Apache2 Server neměl kam poslat *Echo Reply*. Šířka pásma byla zaplavena jen ve směru k zařízení.



Obrázek 14: Síla ICMP flood útoku s podvrženými adresami

Z obrázku [14]můžeme vidět sílu útoku. Síť byla během okamžiku zahlcena pakety. Síla příchozích paketů, podle konzolové aplikace nload, dosahoval síly okolo 68 MBit/s. Z odchozího směru lze ověřit, že počítač opravdu na záplavu paketů nijak nereagoval a vůbec neodpovídal na ping requesty, jelikož útočíme-li s příznakem *–rand-source*, generuje se obrovské množství IP paketů za sekundu na námi stanovený cíl, ale PC odpovídá jen na náš subnet. Na ostatní pakety nereaguje a ignoruje je, jak si můžeme všimnout z výpisu [13]. Naší útočnou silou 5 počítačů jsme sice dokázali zahltit šířku pásma natolik, že volumetrický útok byl úspěšný a dotázání služby, viz. níže, bylo neúspěšné a nebo jen s velkou ztrátou, která tvoří další komunikaci neproveditelnou. Ukážeme si, jak by to vypadalo, kdyby Apache2 Server se pokoušel reagovat na všechny dotazy, které jsme mu poslali.

V síti, připojené k Internetu, by to vypadalo jinak a zařízení by se snažilo reagovat na všechny příchozí ping requesty stejně jako [15].

Vyzkoušeno bylo pingnutí z PC7 na Apache2 Server během útoku a ověření tak jeho dostupnosti.

```
PING 192.168.1.70 (192.168.1.70) 56(84) byte sof data.
64 bytes from 192.168.1.70: icmp_seq=50 ttl=64 time=1012 ms

192.168.1.70 ping statistics
66 packets transmitted, 1 received, 98% paket loss, time 65494ms
```

Z výpisu vidím, že útok byl úspěšný, zahlcení sítě bylo natolik velká, že neprošel téměř žádný ping na Apache2 Server a při dalších pokusech se jevil jako nedostupný. Ping z PC5.

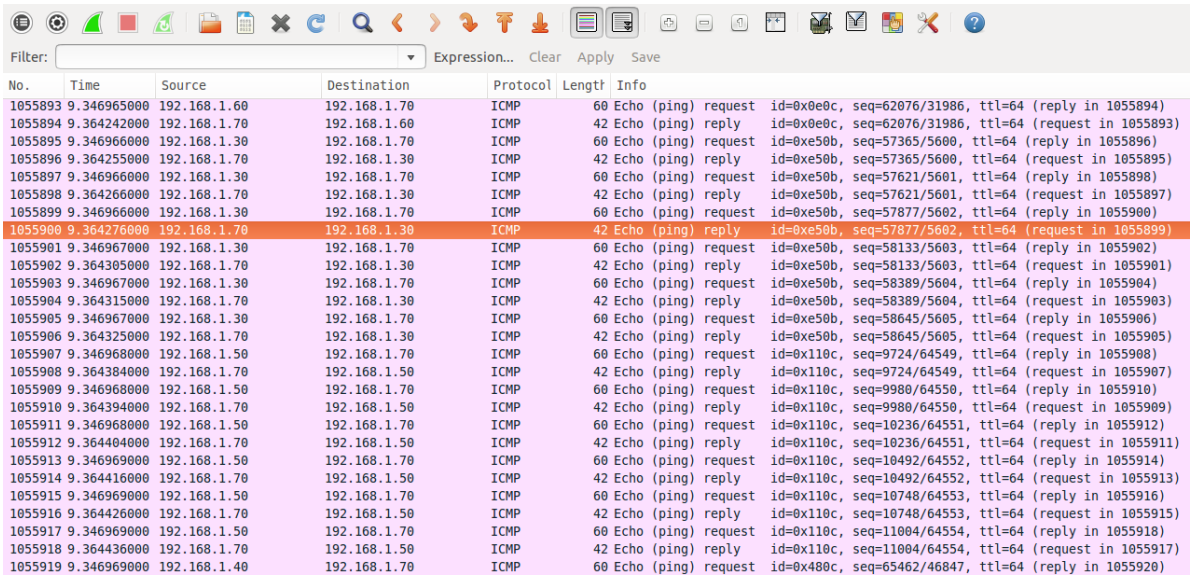
```
root@eb215-desktop:/home/student# ping 192.168.1.70
PING 192.168.1.70 (192.168.1.70) 56(84) byte sof data.
64 bytes from 192.168.1.70: icmp_seq=5 ttl=64 time=7.51 ms
64 bytes from 192.168.1.70: icmp_seq=7 ttl=64 time=7.57 ms
From 192.168.1.70 icmp_seq=9 Destination Host Unreachable
From 192.168.1.70 icmp_seq=9 Destination Host Unreachable
From 192.168.1.70 icmp_seq=9 Destination Host Unreachable
From 192.168.1.70 icmp_seq=9 Destination Host Unreachable
From 192.168.1.70 icmp_seq=9 Destination Host Unreachable

192.168.1.70 ping statistics
17 packets transmitted, 2received, 88% paket loss, time 16041ms
```

Připojení na Apache2 Server poté z PC5 a PC7 vypadal stejně, jako v obrázku [22].

Ukážeme si tedy, jak vypadá útok se statickými adresami z našeho subnetu při záplavovém útoku. Počítač bude reagovat na všechny dotazy a měl by mít snahu na všechny dotazy odpovědět. Pro útok bude zvolen příkaz bez příznaku *–rand-source*, který generuje podvržení adres:

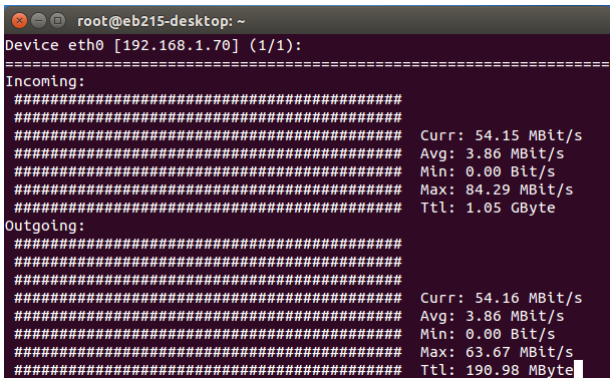
```
hping3 --flood -1 192.168.1.70
```

Výpis z Wiresharku pak vypadal takto:



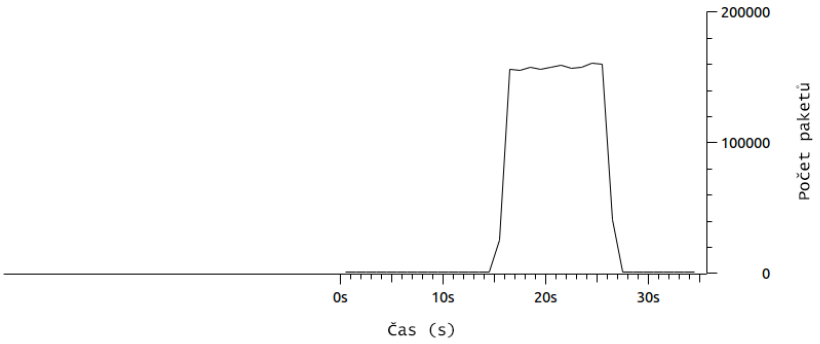
Obrázek 15: ICMP flood s adresami ze sítě - výpis z Wiresharku

Jak vidíme na výpisu [15], útočíme-li na počítač ze svého subnetu, počítač odpovídá na každý dotaz. Překvapující bylo vytížení procesorů na Apache2 Server, které i přes zapojení 5 a chvilkově i více počítačů, nepřesahovalo více jak 15%.



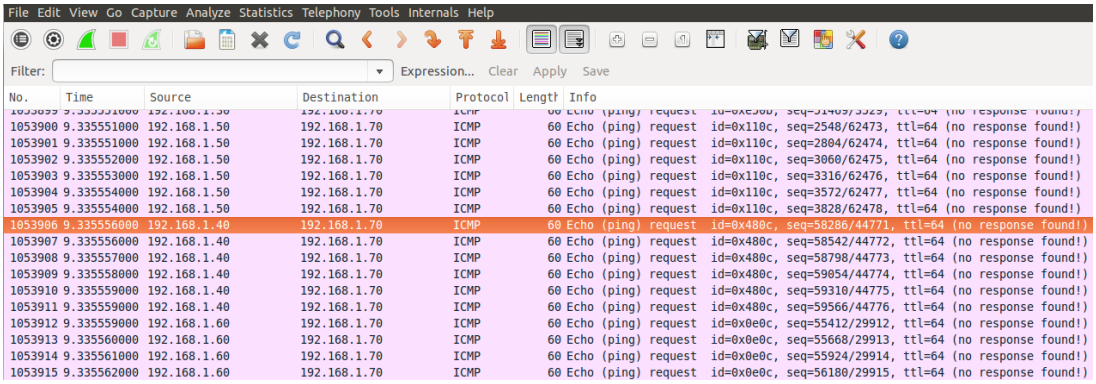
Obrázek 16: Síla ICMP flood útoku s adresami ze sítě

Jak vidíme z obrázku [16], útok dosahoval něco okolo 54 MBit/s v obou směrech. V obrázku [16] na rozdíl od obrázku [14], Apache2 Server na dotazy odpovídal, což je možné vidět z odchozího provozu, a zahlcoval tak ještě více svoji síť. Síť byla zcela zahlcena jak ve směru downlink, tak uplink a byla znemožněna jakákoliv komunikace s Apache2 Server. Chvilkové připojení více počítačů zapřičinilo pád Bridge na PC4 a úplné znemožnění práce se strojem.



Obrázek 17: Počet vygenerovaných paketů za sekundu u ICMP flood útoku

Na obrázku [17] můžeme vidět, že počet paketů při ICMP útoku s nástrojem *hping3* dosahuje přes 150 tisíc paketů za sekundu. Apache2 Server je tedy doslova zaplaven pakety, na které ale ne vždy dovede odpovědět, i když adresa, která provádí ping je z jeho sítě. Obrázek [17]slouží jen pro ukázku množství vygenerování paketů za sekundu programem hping3. Delší zaznamenání bylo znemožněno pádem stroje vlivem puštěného paketového analyzátoru, který tak velký počet příchozích paketů zachycoval a zaměstnal tak práci CPU jednotek natolik, že stroj nebyl schopný provozu a byl vynucen restart.



Obrázek 18: ICMP flood útok - výpis z Wiresharku

Z obrázku [18] lze vyčíst, že i když se ve výpisu dotazovaly počítače ze sítě IP adresy *192.168.1.40*,*192.168.1.50* a *192.168.1.60*, byl to již více než miliontý paket za pár sekund. Pro Apache2 Server je pak z fyzického hlediska legitimně odpovědět na všechny dotazy takřka nemožné.

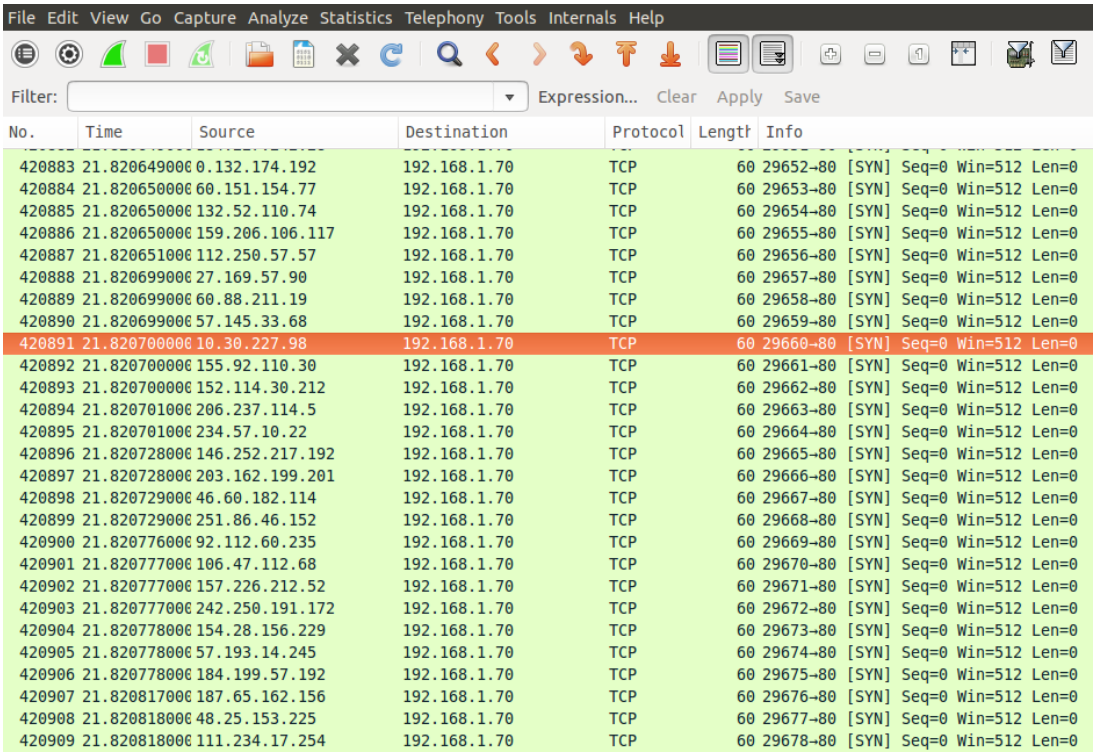
Delší simulace s takovýmto množstvím paketů se ukázala jako nereálná, protože počítače v učebně nedokázaly takovýto přísun paketů z jednoho, natož z více počítačů zvládnout, a takovéto množství paketů bylo ukázkou jen možností programového nástroje hping3.

7.3 TCP SYN flood

Pro útok byl použit na všech útočících zařízeních příkaz:

```
hping3 -S --flood --p 80 --rand-source 192.168.1.70
```

Na cílový stroj Apache2 Server budeme posílat příznaky SYN pomocí *-S* ve výpisu a budeme se dotazovat na port 80. Výpis z Wiresharku s příznakem *–rand-source*



No.	Time	Source	Destination	Protocol	Length	Info
420883	21.820649000	0.132.174.192	192.168.1.70	TCP	60	29652→80 [SYN] Seq=0 Win=512 Len=0
420884	21.820650000	60.151.154.77	192.168.1.70	TCP	60	29653→80 [SYN] Seq=0 Win=512 Len=0
420885	21.820650000	132.52.110.74	192.168.1.70	TCP	60	29654→80 [SYN] Seq=0 Win=512 Len=0
420886	21.820650000	159.206.106.117	192.168.1.70	TCP	60	29655→80 [SYN] Seq=0 Win=512 Len=0
420887	21.820651000	112.250.57.57	192.168.1.70	TCP	60	29656→80 [SYN] Seq=0 Win=512 Len=0
420888	21.820699000	27.169.57.90	192.168.1.70	TCP	60	29657→80 [SYN] Seq=0 Win=512 Len=0
420889	21.820699000	60.88.211.19	192.168.1.70	TCP	60	29658→80 [SYN] Seq=0 Win=512 Len=0
420890	21.820699000	57.145.33.68	192.168.1.70	TCP	60	29659→80 [SYN] Seq=0 Win=512 Len=0
420891	21.820700000	10.30.227.98	192.168.1.70	TCP	60	29660→80 [SYN] Seq=0 Win=512 Len=0
420892	21.820700000	155.92.110.30	192.168.1.70	TCP	60	29661→80 [SYN] Seq=0 Win=512 Len=0
420893	21.820700000	152.114.30.212	192.168.1.70	TCP	60	29662→80 [SYN] Seq=0 Win=512 Len=0
420894	21.820701000	206.237.114.5	192.168.1.70	TCP	60	29663→80 [SYN] Seq=0 Win=512 Len=0
420895	21.820701000	234.57.10.22	192.168.1.70	TCP	60	29664→80 [SYN] Seq=0 Win=512 Len=0
420896	21.820728000	146.252.217.192	192.168.1.70	TCP	60	29665→80 [SYN] Seq=0 Win=512 Len=0
420897	21.820728000	203.162.199.201	192.168.1.70	TCP	60	29666→80 [SYN] Seq=0 Win=512 Len=0
420898	21.820729000	46.60.182.114	192.168.1.70	TCP	60	29667→80 [SYN] Seq=0 Win=512 Len=0
420899	21.820729000	251.86.46.152	192.168.1.70	TCP	60	29668→80 [SYN] Seq=0 Win=512 Len=0
420900	21.820776000	92.112.60.235	192.168.1.70	TCP	60	29669→80 [SYN] Seq=0 Win=512 Len=0
420901	21.820777000	106.47.112.68	192.168.1.70	TCP	60	29670→80 [SYN] Seq=0 Win=512 Len=0
420902	21.820777000	157.226.212.52	192.168.1.70	TCP	60	29671→80 [SYN] Seq=0 Win=512 Len=0
420903	21.820777000	242.250.191.172	192.168.1.70	TCP	60	29672→80 [SYN] Seq=0 Win=512 Len=0
420904	21.820778000	154.28.156.229	192.168.1.70	TCP	60	29673→80 [SYN] Seq=0 Win=512 Len=0
420905	21.820778000	57.193.14.245	192.168.1.70	TCP	60	29674→80 [SYN] Seq=0 Win=512 Len=0
420906	21.820778000	184.199.57.192	192.168.1.70	TCP	60	29675→80 [SYN] Seq=0 Win=512 Len=0
420907	21.820817000	187.65.162.156	192.168.1.70	TCP	60	29676→80 [SYN] Seq=0 Win=512 Len=0
420908	21.820818000	48.25.153.225	192.168.1.70	TCP	60	29677→80 [SYN] Seq=0 Win=512 Len=0
420909	21.820818000	111.234.17.254	192.168.1.70	TCP	60	29678→80 [SYN] Seq=0 Win=512 Len=0

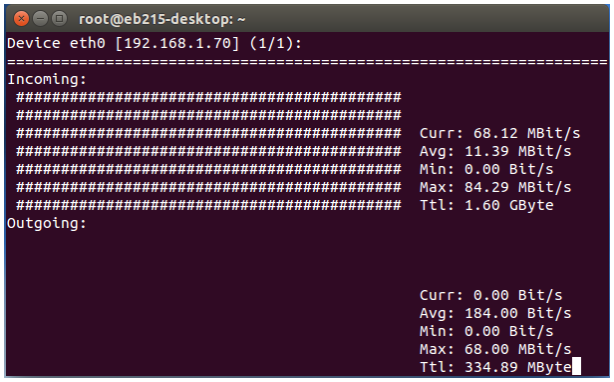
Obrázek 19: TCP SYN flood s podvrženými adresami - výpis z Wiresharku

Tak jako v předešlém útoku ICMP flood je možno vidět to samé i tady, jen s protokolem TCP, který se snaží navázat spojení na portu 80. Útok generoval podvržené adresy v řádech statisíců za sekundu.

Výsledek byl opět stejný, s počítačem nebylo možné komunikovat, a to zejména díky množství paketů vyslaného k Apache2 Server, ne však díky samotné podstatě útoku, kterou měl TCP SYN flood útok představovat. Jak můžeme vidět na obrázku [19], generované IP adresy se snažily navázat spojení s Apache2 Server pomocí poslání příznaku SYN na portu 80, ale neúspěšně. Apache2 Server nereagoval na příchozí pakety.

Tímto způsobem bylo docíleno jen efektu volumetrického útoku, který zaplavil síť množstvím paketů, jako tomu bylo u ICMP flood u příkladu výše. Síť byla zahlcena pakety ze stran počítačů, ale nebylo úspěšně docíleno kýženého TCP SYN floodu.

Síla útoku dosahovala podobných hodnot jako předešlý útok. A z odchozího provozu si lze opět ověřit, že počítač vůbec na příchozí pakety nereagoval.



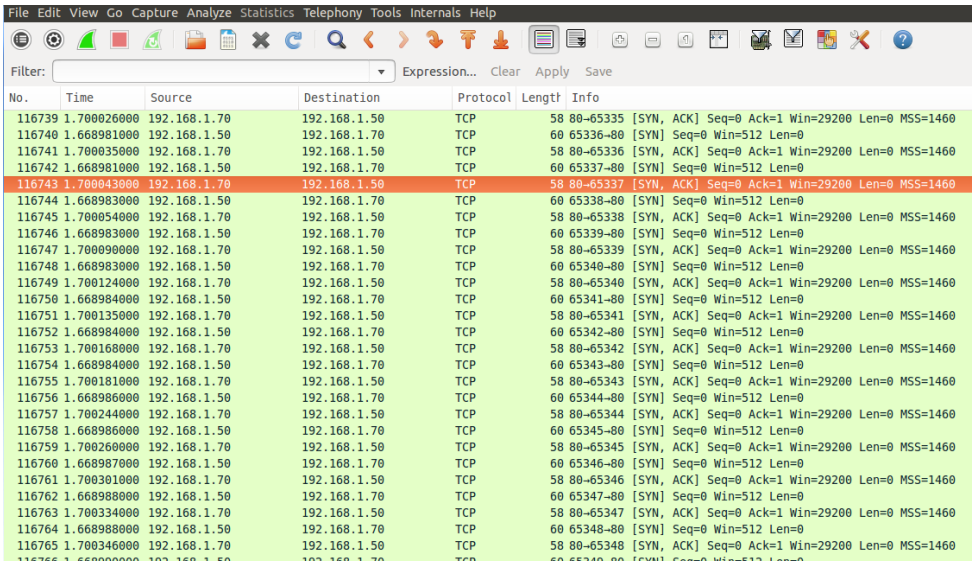
Obrázek 20: Síla TCP SYN flood útoku s podvrženými adresami

Z předešlého případu víme, že zkusíme-li navázat spojení z adres z našeho subnetu, Apache2 Server bude odpovídat na dotazy a útok bude útok efektivní.

Provedeme útok SYN flood pomocí příkaz:

```
hping3 -S --flood --p 80 192.168.1.70
```

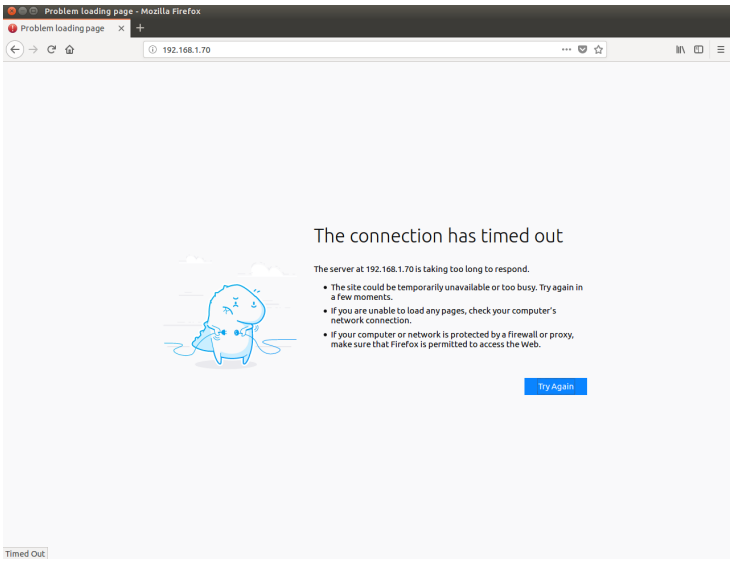
Výpisu z Wiresharku při útoku SYN flood pak vypadal takto:



Obrázek 21: TCP SYN flood s adresami ze sítě - výpis z Wiresharku

Z Wireshark výpisu [21] jasně vidíme snahu navázat spojení zasláním příznaku *SYN,ACK* z Apache2 Server k jednomu z útočících PC. Apache2 Server měl snahu o navázání Three-way handshake. Útočící stroj poslal příznak SYN na který mu Apache2 Server odpovídalo příznaky SYN, ACK čekajíc na odpověď ACK ze strany útočícího stroje, která, jak vidíme v obrázku[21], nepřicházela a bylo takhle navázáno, jak je v teoretické části zmíněno, napůl otevřené spojení.

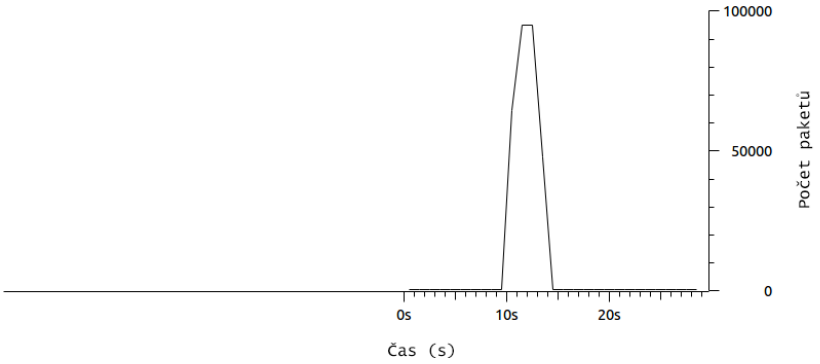
Pokus o připojení z PC5 na Apache2 Server.



Obrázek 22: Nedostupnost stránky na Apache2 Serveru

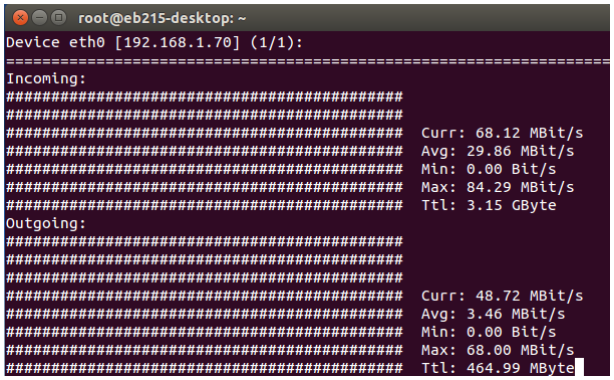
Výsledek ověřil úspěšnost útoku. Stránku nebylo možné ani po několika pokusech načíst. Apache2 Server měl otevřeno příliš mnoho spojení. Pochyby o probíhajícím útoku, bylo možné pozorovat hned z počátku, stránka načítala strašně pomalu a od první chvíle bylo zřejmé, že je něco špatně, až se vyhodila chybová hláška.

Stránku z 5 pokusů nebylo možné načíst ani jednou. Tím to byl TCP SYN flood úspěšný. Na obrázku lze pak vidět, že počet odeslaných paketů byl jiný, než tomu bylo u ICMP flood útoku. U TCP SYN floodu dokáže program Hping3 generovat okolo 100 tisíc za sekundu.



Obrázek 23: Počet vygenerovaných paketů za sekundu u TCP SYN flood útoku

Síla útoku byla stejná a Apache2 Server reagoval na pakety a snažil se navázat three-way handshake.



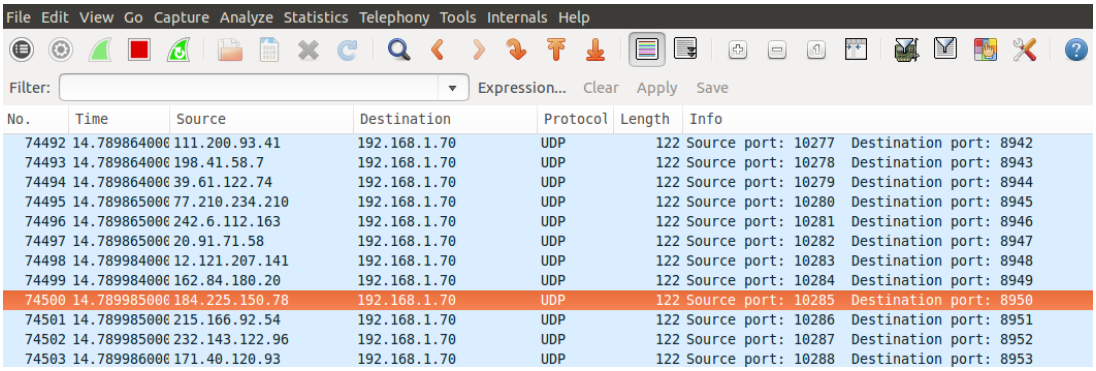
Obrázek 24: Síla TCP SYN flood útoku s adresami ze sítě

7.4 UDP flood

Pro útok byl použit na všech útočících zařízeních příkaz:

```
hpin3 --flood -2 --p ++1 --d ++1 --rand-source 192.168.1.70
```

Výpis z Wiresharku:



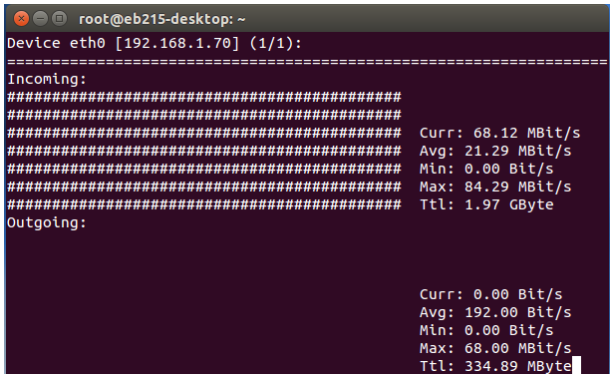
Obrázek 25: UDP flood s podvrženými adresami - výpis z Wiresharku

Opět z výpisu [25] vidíme, že Apache2 Server obdržel statisíce UDP paketů za pár sekund. Source a destination port se navyšoval po jedné, díky příznakům *–p ++1 –d ++1* v příkazu, aby věrohodně simuloval UDP flood. Služba by v tomto případě měla odpovídat na dotazy IP adres ICMP pakety Destination unreachable.

U UDP floodu je zaslán UDP paket na náhodné porty, v našem případě se navyšuje každý paket o jeden, Apache2 Server by měl zkontrolovat, jestli na portu je nějaká služba, není-li, posílá zpět ICMP paket o nedostupnosti služby. Jak však vidíme, tak stejně jako v předchozích případech, Apache2 Server vůbec nereaguje na UDP pakety a je dosaženo pouze volumetrického útoku, kdy je síť zahlcena pakety.

Volumetrický útok tím znemožnil komunikaci po síti, ale nebylo docíleno UDP floodu a jeho podstaty.

Útok UDP protokolem by byl samozřejmě úspěšný, když zvolíme například službu HTTP s portem 80 a službu zaplavíme obrovským množstvím paketů z více počítačů, docílíme úspěšného volumetrického útoku. Počet UDP paketů odeslaných za sekundu s příznakem *–flood* se přibližně rovnal stejnému počtu, jako tomu bylo u ICMP floodu, tedy něco přes 150 tisíc paketů za sekundu.



Obrázek 26: Síla UDP flood útoku s podvrženými adresami

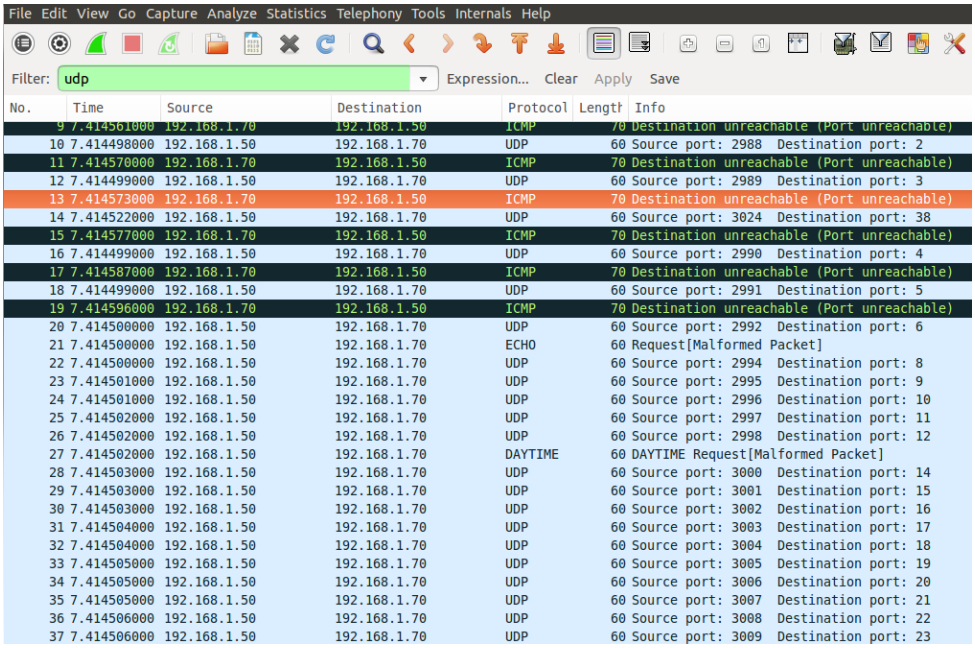
Jak můžeme pozorovat, počítač neměl vůbec snahu reagovat na příchozí UDP pakety.

Vyzkoušíme-li ale opět naše adresy ze sítě, obrázek útoku by se měl změnit a Apache2 Server na útoky reagovat.

Útok jsme vyzkoušeli pomoci příkazu:

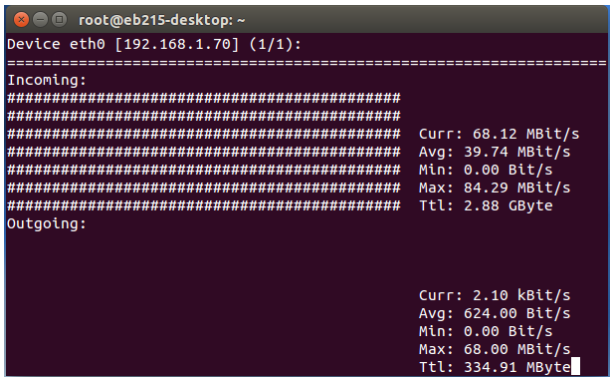
```
hping3 --flood -2 --p ++1 192.168.1.70
```

Vypis z Wiresharku pak vypadal takto:



Obrázek 27: UDP flood s adresami ze sítě - výpis z Wiresharku

Apache2 Server jsme zaplavili UDP pakety na náhodné porty, které můžeme vidět se navyšovaly po jedné, přesně jak jsme chtěli. Avšak z výpisu [27] jsem zjistil, že příznak *–flood* v našem případě nelze použít a to proto, že Apache2 Server nestihá vůbec reagovat na UDP pakety a služba stihla reagovat jen na pár prvních paketů viz. [27]. Tím je docíleno opět pouze volumetrického útoku.

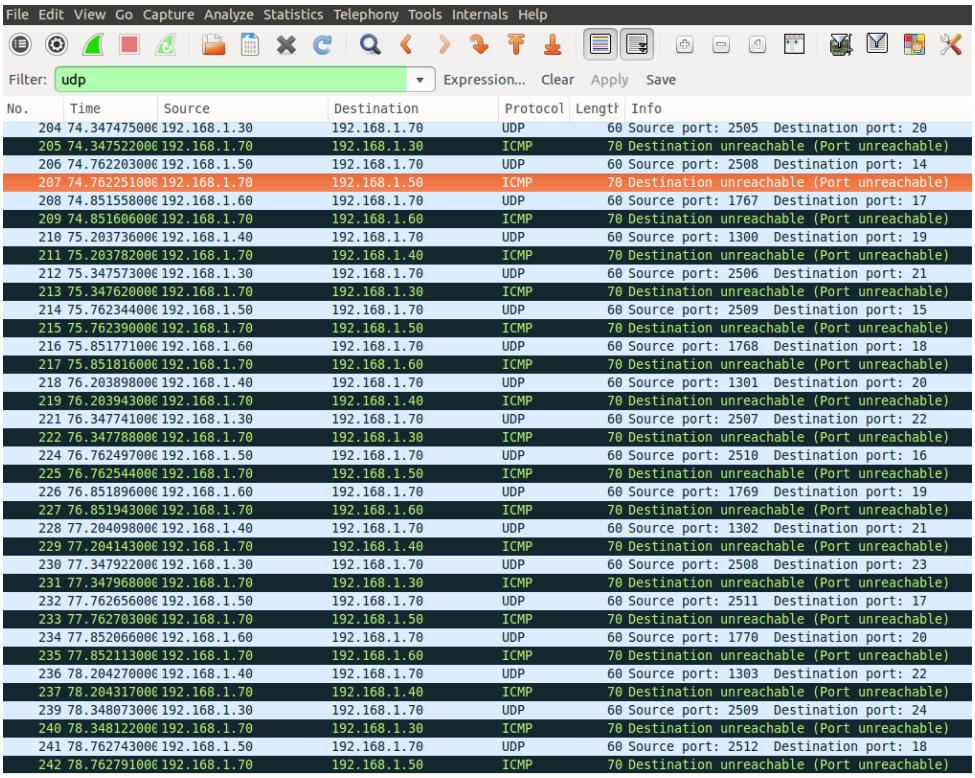


Obrázek 28: Síla UDP flood útoku s adresami ze sítě

Z odchozího provozu vidíme, že počítač reaguje na příchozí pakety ICMP pakety, ale jen ve velmi omezeném množství.

Zkusil jsem tedy UDP flood bez příznaku *–flood*, kdy počítač posílá pomalu UDP pakety na cílový počítač, aby jsme ověřili, jestli i při menším rychlosti zvládne koncový stroj Apache2 Server odpovídat ICMP pakety.

Výpis z Wiresharku poté vypadal takto:



No.	Time	Source	Destination	Protocol	Length	Info
204	74.34747500€	192.168.1.30	192.168.1.70	UDP	60	Source port: 2505 Destination port: 20
205	74.34752200€	192.168.1.70	192.168.1.30	ICMP	70	Destination unreachable (Port unreachable)
206	74.76220300€	192.168.1.50	192.168.1.70	UDP	60	Source port: 2508 Destination port: 14
207	74.76225100€	192.168.1.70	192.168.1.50	ICMP	70	Destination unreachable (Port unreachable)
208	74.85155800€	192.168.1.60	192.168.1.70	UDP	60	Source port: 1767 Destination port: 17
209	74.85160600€	192.168.1.70	192.168.1.60	ICMP	70	Destination unreachable (Port unreachable)
210	75.20373600€	192.168.1.40	192.168.1.70	UDP	60	Source port: 1300 Destination port: 19
211	75.20378200€	192.168.1.70	192.168.1.40	ICMP	70	Destination unreachable (Port unreachable)
212	75.34757300€	192.168.1.30	192.168.1.70	UDP	60	Source port: 2506 Destination port: 21
213	75.34762000€	192.168.1.70	192.168.1.30	ICMP	70	Destination unreachable (Port unreachable)
214	75.76234400€	192.168.1.50	192.168.1.70	UDP	60	Source port: 2509 Destination port: 15
215	75.76239000€	192.168.1.70	192.168.1.50	ICMP	70	Destination unreachable (Port unreachable)
216	75.85177100€	192.168.1.60	192.168.1.70	UDP	60	Source port: 1768 Destination port: 18
217	75.85181600€	192.168.1.70	192.168.1.60	ICMP	70	Destination unreachable (Port unreachable)
218	76.20389800€	192.168.1.40	192.168.1.70	UDP	60	Source port: 1301 Destination port: 20
219	76.20394300€	192.168.1.70	192.168.1.40	ICMP	70	Destination unreachable (Port unreachable)
221	76.34774100€	192.168.1.30	192.168.1.70	UDP	60	Source port: 2507 Destination port: 22
222	76.34778800€	192.168.1.70	192.168.1.30	ICMP	70	Destination unreachable (Port unreachable)
224	76.76249700€	192.168.1.50	192.168.1.70	UDP	60	Source port: 2510 Destination port: 16
225	76.76254400€	192.168.1.70	192.168.1.50	ICMP	70	Destination unreachable (Port unreachable)
226	76.85189600€	192.168.1.60	192.168.1.70	UDP	60	Source port: 1769 Destination port: 19
227	76.85194300€	192.168.1.70	192.168.1.60	ICMP	70	Destination unreachable (Port unreachable)
228	77.20409800€	192.168.1.40	192.168.1.70	UDP	60	Source port: 1302 Destination port: 21
229	77.20414300€	192.168.1.70	192.168.1.40	ICMP	70	Destination unreachable (Port unreachable)
230	77.34792200€	192.168.1.30	192.168.1.70	UDP	60	Source port: 2508 Destination port: 23
231	77.34796800€	192.168.1.70	192.168.1.30	ICMP	70	Destination unreachable (Port unreachable)
232	77.76265600€	192.168.1.50	192.168.1.70	UDP	60	Source port: 2511 Destination port: 17
233	77.76270300€	192.168.1.70	192.168.1.50	ICMP	70	Destination unreachable (Port unreachable)
234	77.85206600€	192.168.1.60	192.168.1.70	UDP	60	Source port: 1770 Destination port: 20
235	77.85211300€	192.168.1.70	192.168.1.60	ICMP	70	Destination unreachable (Port unreachable)
236	78.20427000€	192.168.1.40	192.168.1.70	UDP	60	Source port: 1303 Destination port: 22
237	78.20431700€	192.168.1.70	192.168.1.40	ICMP	70	Destination unreachable (Port unreachable)
239	78.34807300€	192.168.1.30	192.168.1.70	UDP	60	Source port: 2509 Destination port: 24
240	78.34812200€	192.168.1.70	192.168.1.30	ICMP	70	Destination unreachable (Port unreachable)
241	78.76274300€	192.168.1.50	192.168.1.70	UDP	60	Source port: 2512 Destination port: 18
242	78.76279100€	192.168.1.70	192.168.1.50	ICMP	70	Destination unreachable (Port unreachable)

Obrázek 29: UDP flood s adresami ze sítě - výpis z Wiresharku

Vidíme, že při pomalejším odesílání paketu UDP, Apache2 Server reagoval na odeslané pakety ICMP odpovědí přesně podle toho, jak je popsáno v teoretické části. Na obrázku [29] lze vidět, jak Apache2 Server reagoval na každý jednotlivý dotaz.

8 Anti-DDoS program

8.1 Spuštění programu na Ubuntu 14.04

V této sekci si popíšeme kroky potřebné k úspěšnému spuštění DDoS programu na Ubuntu 14.04, ale nejen na něm, chceme-li aplikovat program na jiných OS, kroky budou obdobné. Proto byl C# .NET Core vybrán pro implementaci, aby plnil v maximální míře flexibilitu open-source programu s jen případnými drobnými úpravami kódu na různých OS.

Program C# .NET Core je tedy možné spustit na zařízeních využívající operační systém Linux, pomocí kroků, které musí být dodrženy v daném pořadí, aby bylo možné s programem v prostředí Ubuntu pracovat.

Prvním krokem je stažení MonoDevelop ve kterém můžeme samotnou aplikaci vytvářet a spouštět. Příkazy k instalaci klíčů a všech potřebných věcí na všech OS jsou k dispozici na oficiálních stránkách MonoDevelop.[12]

Pomocí příkazů na těchto stránkách[12] bylo staženo a nainstalováno vývojové prostředí MonoDevelop, k tomu musíme doinstalovat .NET Core, který je možno nainstalovat pomocí příkazů a návodů ze stránek Microsoftu[13]. Je nezbytnou součástí instalace přesně dodržet kroky v daném pořadí a úspěšně nainstalovat všechny potřebné věci ke spuštění programu.

Uvedenými kroky na stránkách[13] bylo úspěšně nainstalován .NET Core na PC4. Už je potřeba jen doinstalovat libPcap, protože .NET Core pracuje s wpcap, což funguje jen pod Windows OS, je potřeba tedy nainstalovat libPcap, který funguje pod Linuxem.

A jako poslední krok musíme stáhnout ze stránky[14] GitHubu knihovnu z níž .NET Core pod Linuxem, ale taky pod jinými OS, pracuje. Tato knihovna musí být naimportována do projektu, dochází v ní k přemapování wpcap na libpcap, používání knihoven PacketDotNet a SharpPcap, je tedy nezbytná pro úspěšný spuštění programu pod Linuxem, ale taky pod jakýmkoliv OS, ať už je to Windows, Mac OS X, nebo dalších.

Výše zmíněnými kroky je připraveno vývojové prostředí pro spuštění. Poté byl nastaven PC4 jako Bridge a dokonfigurován zbytek sítě.

8.2 Logika a funkce programu

Cílem této části je seznámení s logikou a funkcí vytvořeného open-source systému pro detekci a potlačení DDoS útoků.

Program je spuštěn na PC4 a odchytává pakety s rozpoznáváním typu provozu, směřujících od S1 k S2, přes PC4 po eth2 k Apache2 Serveru.

Základem programu je počet přijatých paketů za sekundu z určitého typu provozu, v našem případě TCP, UDP nebo ICMP. Přesáhne-li počet přijatých paketů stanovenou mez, provede se jedna z podmínek programu na omezení, nebo úplné potlačení útoku.

Program tedy počítá počet přijatých paketů za sekundu jdoucích po eth2. Při spuštění programu zadáváme sedm parametrů pro každý provoz. Dohromady jich tedy zadáváme před kaž-

dým spuštěním programu 21 parametrů. Takovéto množství proměnných je zvoleno proto, aby bylo docíleno co největší flexibility programu na určité infrastrukturní lince, na které je program spuštěn.

Následná posloupnost parametrů je následující:

```
Maximální počet paketů za sekundu, po které bude provoz omezen:
Maximální počet paketů na které bude provoz omezen:
Čas po kterou bude provoz omezen:
Maximální počet paketů po kterém bude síť nedostupná:
Čas po která bude síť nedostupná:
Maximální počet paketů za sekundu z jedné IP adresy:
Čas po který bude IP adresa bloknutá:
```

Tyto parametry zadáváme se spuštěným program za sebou pro každý typ provozu. Začíná se u TCP, poté UDP a nakonec ICMP. Po každém provedení akce programem se do konzole vypíše akce, která byla provedena. Vypsán bude vždy jen ten typ provozu, kterého se dané dosažené parametry týkají.

První tři parametry nám určují, že při dosažení daného počtu paketů za sekundu, se na námi stanovenou dobu omezí počet paketů, jdoucí skrz PC4 na námi stanovenou hodnotu a čas pro TCP, UDP a ICMP.

Při provedení této akce se nám do konzole vypíše hláška:

```
Byl detekovan hranicni pocet "zadaný parametr" TCP/UDP/ICMP paketu za sekundu.
TCP/UDP/ICMP provoz bude omezen na "zadaný parametr" paketu za sekundu na dobu
"zadaný parametr" sekund.
```

Po uplynutí zadaného časového parametru, který je při prvních třech parametrech *Čas po kterou bude provoz omezen:* bude vypsána hláška o ukončení omezení provozu:

```
TCP/UDP/ICMP provoz po omezeni obnoven.
```

Třetí a čtvrtý parametr nám určuje počet paketů za sekundu, při kterém má být veškerý provoz na rozhraní zahozen na námi stanovenou hodnotu a čas. Tuto možnost jsem zvolil, neboť počet paketů za sekundu může vzrůst o nadměrné množství, například připojením většího počtu počítačů do útoku, nebo navýšením vyslaných počtu paketů z jednoho z PC a zesílením útoku. Program automaticky vyhodnotí jako pokus o probíhající DDoS útok a veškeré příchozí pakety po rozhraní budou na zadaný čas zahozeny. Ochrání tak síť nebo zařízení, před kterým filtruje provoz, od útoku.

Při provedení této akce se nám do konzole vypíše hláška:

```
Byl detekovan maximalni pocet "zadany parametr" TCP/UDP/ICMP paketu za sekundu.
Sit bude na "parametr" sekund nedostupna - Podezreni na probihajici DDoS utok.
```

Po uplynutí zadaného časového parametru, který jsme zadali, se nám vypíše hláška :

```
TCP/UDP/ICMP provoz obnoven.
```

Poslední šestý a sedmý parametr, který zadáváme u daného typu provozu, je počet paketů za sekundu, po kterém bude veškerý provoz zahozen jen z jednotlivé IP adresy. Při našich simulacích totiž můžeme využít příznak *–rand-source*, který generuje velké množství náhodných IP adres, avšak bez tohoto příznaku může být vysláno větší množství paketů z jedné IP adresy, než je hraniční parametr. Můžeme tedy definovat počet paketů z jednotlivé adresy, který se nám zdá nadměrný a vyhodnotit to, jako pokus o probíhající útok, program opět automaticky vyhodnotí, jako pokus o DoS/DDoS útok a provoz z dané IP adresy zakáže. Dosáhneme tak daleko efektivnějšího řešení, protože další provoz nemusí být nijak ovlivněn.

Při provedení této akce se nám do konzole vypíše hláška:

```
Byl detekovan maximalni pocet "zadany parametr" TCP/UDP/ICMP paketu z jedne IP adresy
za sekundu.
Adresa IP je zablokovana na "zadany parametr" sekund.
```

Po uplynutí zadaného časového parametru se vypíše:

```
Adresa IP je odblokovana.
```

Zadáním těchto proměnných je dána uživateli flexibilita v používání na různých infrastrukturních linkách, kde může být počet paketů tekoucích po lince za sekundu různý.

Při provedení určité operace program vypíše, co bylo provedeno za akci, na jaký počet paketů byl provoz omezen a na jak dlouho, jak uvidíme u simulacích.

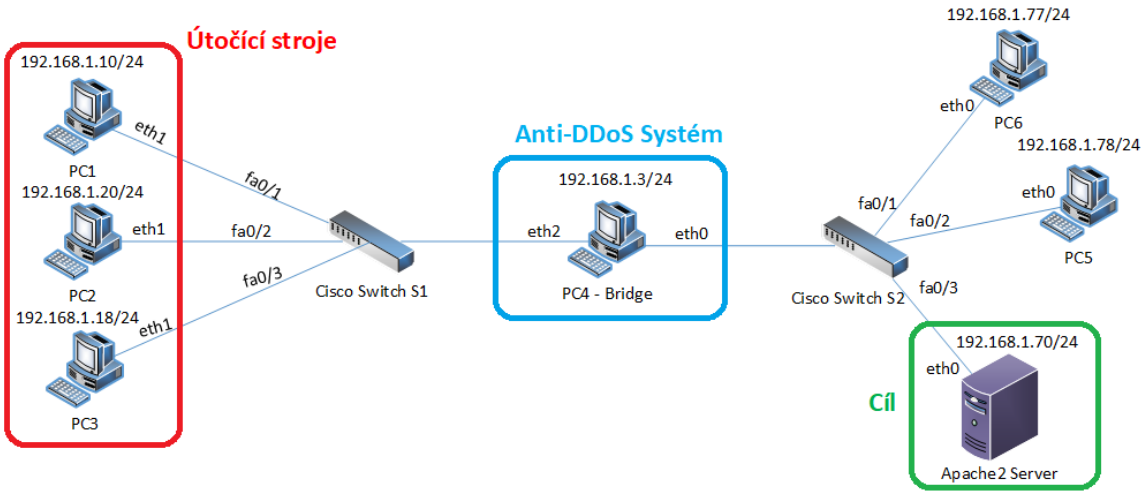
8.3 Testování programu v laboratorním prostředí

Pro testování vytvořeného programu bylo nutné omezit počet paketů vyslaných k Apache2 Server, upravení topologie a znázornění funkce a efektivity programu v menším měřítku. Počítače v učebně eb215 nebyly schopny pracovat s tak velkým počtem vyslaných paketů. Při spuštění dvou paketových analyzátorů na PC4, spuštěném Anti DDoS programu a přitom jeho filtrování, i při zlomkovém počtu paketů docházelo po pár sekundách k zamrznutí PC4, a to vlivem výpočetní síly počítače, veškerá práce, instalace a další nutné úkony pro úspěšné otestování, bylo nutné opakovat.

Po konzultaci toho faktu s vedoucím práce, byla zvolena práce s menším počtem paketů pro demonstrativní ukázku funkčnosti programu. Síla útoků tedy bude dosahovat daleko menších hodnot, než tomu bylo v sekci Simulace útoku, avšak o to bude lépe vidět práce programu.

Názornou ukázkou funkčnosti programu bude tedy rozdíl počtu paketů po úspěšném filtrování paketů jdoucích z eth2 na eth0 z PC1-3 přes PC4 k Apache2 Server.

8.3.1 Topologie



Obrázek 30: Topologie pro testování Anti DDoS programu

Topologie na obrázku[57] byla použita u všech simulací UDP, ICMP, tak i TCP.

8.3.2 UDP flood

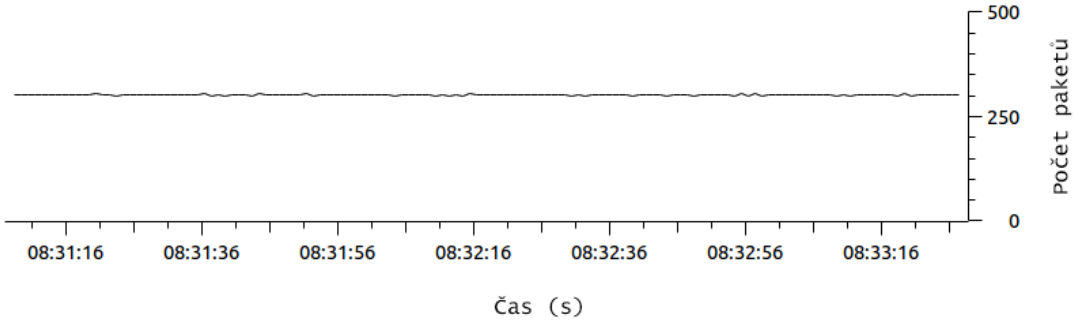
K útoku byl využit program hping3 u kterého bylo nastaveno počet vyslaných paketů za sekundu na každém útočícím počítači. Pomocí příznaku *–fast* v příkazu byla zmenšena frekvence a *-i u20000* nám definuje počet vyslaných paketů, který se pohyboval okolo 80 až 90 za sekundu, u PC2 byl nastaven příkaz s příznakem *–rand-source*.

```
hping3 --fast -2 -i u20000 192.168.1.70
```

Na takový tok paketů byl aplikován Anti-DDoS program s hraničními parametry:

```
Maximální počet paketů za sekundu, po které bude provoz omezen: 250
Maximální počet paketů na které bude provoz omezen: 50
Čas po kterou bude provoz omezen: 10
Maximální počet paketů po kterém bude síť nedostupná: 600
Čas po která bude síť nedostupná: 20
Maximální počet paketů za sekundu z jedné IP adresy: 700
Čas po který bude IP adresa bloknutá: 10
```

U času jsou použity parametry v sekundách. Po následném nasazení programu na provoz vypadal provoz následovně.



Obrázek 31: UDP flood - Příchozí paketový tok na PC4 - eth2

Po spuštění Anti-DDoS programu, program zachytil hraniční počet parametrů, která spadala do podmínky programu na omezení provozu:

```
if (currentTraffic > UDPMaximumTrafficPerSecond && completeShutDownTCP == false
    ) {
  var ruleName = _random.Next (0, 999999).ToString ();
  Console.WriteLine ($"Byl detekovan hranicni pocet
  {UDPMaximumTrafficPerSecond} UDP paketu za sekundu. " + $"UDP provoz bude
     omezen na {UDPMaximumTrafficPacketPerSecond} paketu za sekundu na dobu {
     UDPMaximumTrafficReductionDuration} sekund.");
```

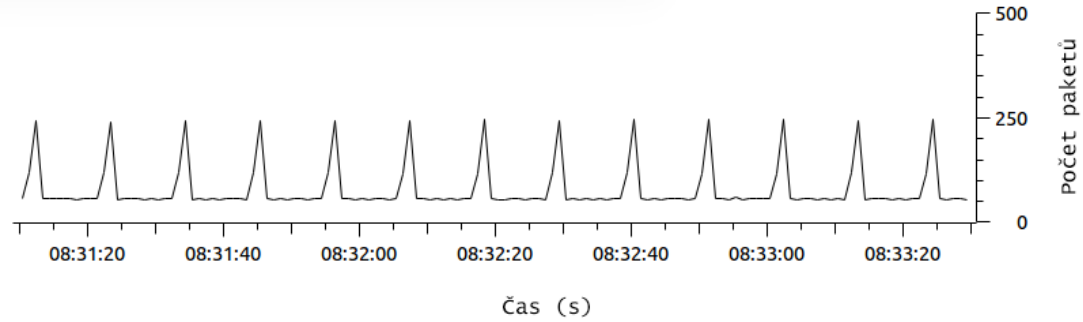
Výpis 1: C# .NET Core na Ubuntu 14.04. Podmínka na omezení UDP provozu

Po splnění této podmínky se přímo vykoná příkaz s ebtables na redukci provozu dle parametru *UDPMaximumTrafficPacketPerSecond* zadaného na začátku programu.

```
  ExecuteCommand ($"ebtables -N {ruleName};" +
     $"ebtables -A FORWARD -p IPv4 --ip-protocol udp -j {ruleName};" +
    $"ebtables -A {ruleName} -p IPv4 --limit {UDPMaximumTrafficPacketPerSecond
       }/second  --limit-burst 3 -j ACCEPT;" + $"ebtables -A {ruleName} -j DROP
       ");
  var timer = new Timer (MaximumTrafficUDPPerSecond_Timer, ruleName,
   UDPMaximumTrafficReductionDuration * 1000, Timeout.Infinite);
}
```

Výpis 2: C# .NET Core na Ubuntu 14.04. Ebtables příkazy v programu na omezení UDP provozu

Po splnění podmínek na omezení UDP provozu, se tok paketů z PC4 zredukuje na zadané množství *UDPMaximumTrafficPacketPerSecond*, jak můžeme vidět z odchozího paketového grafu [32].



Obrázek 32: UDP flood - Odchozí paketový tok z PC4 po omezení - eth0

Do konzole bylo programem vypsáno:

```
Byl detekovan hranicni pocet 250 UDP paketu za sekundu.
UDP provoz bude omezen na 50 paketu za sekundu na dobu 10 sekund.
```

A po uplynutí 10 sekund byl příkaz zrušen programem pomocí příkazu

```
static void MaximumTrafficUDPPerSecond_Timer (Object stateInfo)
        {
            Console.WriteLine ($"Provoz po omezeni UDP provozu obnoven");
            var ruleName = stateInfo.ToString ();
            ExecuteCommand ($"ebtables -D FORWARD -p IPv4 --ip-protocol udp -j {
                ruleName};" +
                           $"ebtables -D {ruleName} -p IPv4 --limit {
                               ICMPMaximumTrafficPacketPerSecond}/second --limit-
                               burst 3 -j ACCEPT;" +
                           $"ebtables -D {ruleName} -j DROP;" +
                           $"ebtables -X {ruleName};");
        }
```

Výpis 3: C# .NET Core na Ubuntu 14.04. Smazání ebtables pravidel na omezení UDP provozu

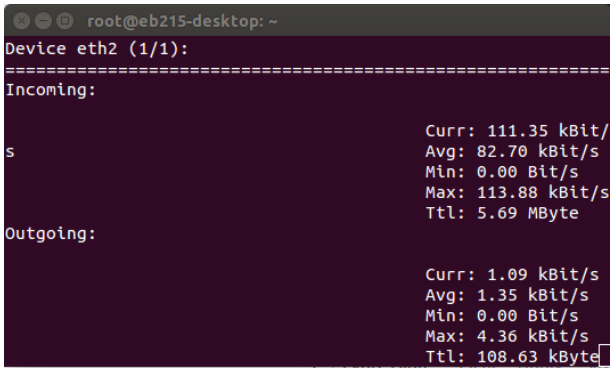
Do konzole bylo programem vypsáno:

```
Provoz UDP po omezení obnoven.
```

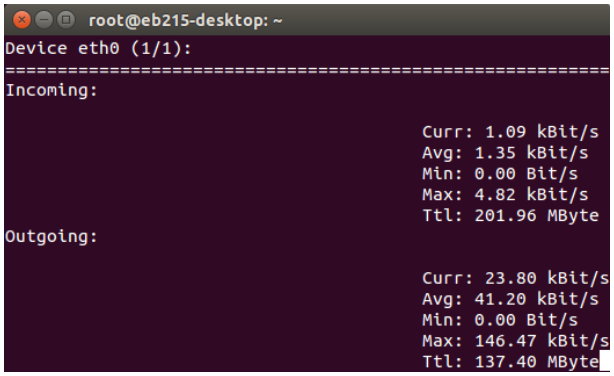
Avšak byl znova detekován hraniční počet paketů udaný parametrem a tak se periodicky opakovala redukce paketů, dokud počet paketů nebude nižší, než je hraniční hodnota.

Na obrázku číslo[31] lze pozorovat, že počet příchozích paketů k PC4 byl pořád stejný a neměnil se, avšak odchozí provoz z PC4 už byl díky programu modifikován, každých 10 sekund byl provoz omezen vlivem následného dosažením limitu 250 paketů za sekundu, jak můžeme vidět na obrázku [32]. Tímto byl UDP flood periodicky omezován.

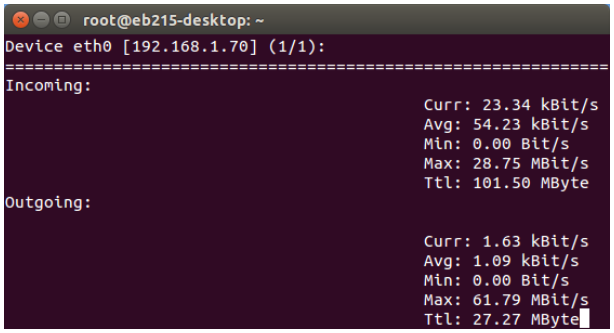
Jak lze vidět na obrázcích z [33] na PC4, tok proudících paketů k PC4 se neměnil, avšak odchozí provoz už dosahoval síly okolo 23 kBit/s, jak dokazují obrázky z [34] na PC4 a Apache2 Server [35].



Obrázek 33: UDP flood - Příchozí paketový tok na PC4 - eth2

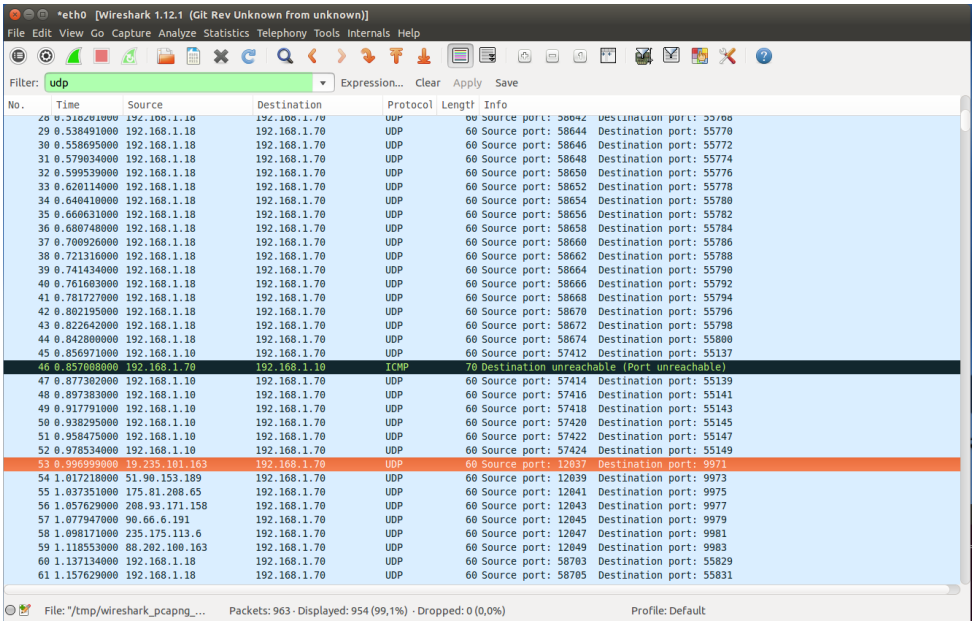


Obrázek 34: UDP flood - Odchozí paketový tok z PC4 - eth0



Obrázek 35: UDP flood - Příchozí paketový tok na Apache2 Server - eth0

Tím byla potvrzena funkčnosti Anti-DDoS programu pro omezení provozu. Propustnost parametru omezení 50 paketů za sekundu si můžeme ověřit na Apache2 Server v paketovém analyzátoru Wireshark, viz. obrázek [36].



Obrázek 36: UDP flood: Příchozí paketový tok na Apache2 Server - výpis z Wiresharku

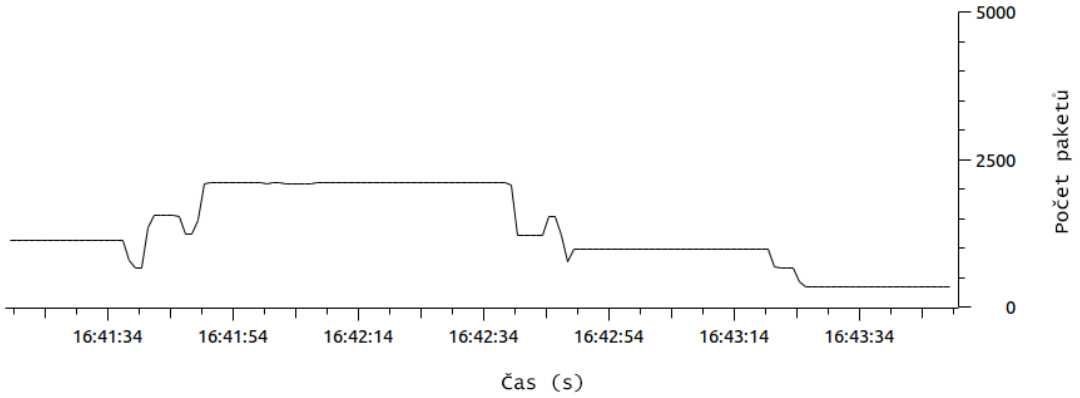
Z obrázku [36]. je zřejmé, že za sekundu provozu bylo přijato okolo UDP 53 paketů. Zadány byly větší hodnoty a vyzkoušeno bylo odstřihnutí ethernetové linky eth2. Parametry byly zadány:

```
Maximální počet paketů za sekundu, po které bude provoz omezen: 1000
Maximální počet paketů na které bude provoz omezen: 200
Čas po kterou bude provoz omezen: 10
Maximální počet paketů po kterém bude síť nedostupná: 2000
Čas po která bude síť nedostupná: 20
Maximální počet paketů za sekundu z jedné IP adresy: 1100
Čas po který bude IP adresa bloknutá: 10
```

K útoku byl využit příkaz:

```
hping3 --fast -2 -i u5000 --randsource 192.168.1.70
```

Z grafu [37] lze vidět, že počet paketů vyslaných k počítači dosahoval kolem 1000 paketů za sekundu, přičemž se plnila ta samá podmínka a výpisy co byly popsány výše.



Obrázek 37: UDP flood: Příchozí paketový tok na PC4 - eth2

V čase 16:41:50 na časové ose, byl navýšen počet paketů na maximální povolený počet. Pomocí stejného příkazu jen změněním parametru *u2000*.

Počet paketů vyslaných počítači k Apache2 Server byl programem vyhodnocený jako maximální možný počet paketů, pro který bude síť dostupná. Splní se první podmínka pro UDP provoz a to:

```
if (currentTraffic > UDPMaximumTrafficToShutDownPerSecond) {
   Console.WriteLine ($"Byl detekovan maximalni pocet
   {UDPMaximumTrafficToShutDownPerSecond} UDP paketu za sekundu. " +
    $"Sit bude na {UDPMaximumTrafficToShutDownDuration} sekund nedostupna -
        Podezreni na probihajici DDoS utok");
    completeShutDownUDP = true;
```

Výpis 4: C# .NET Core na Ubuntu 14.04. Podmínka pro maximální počet UDP provozu

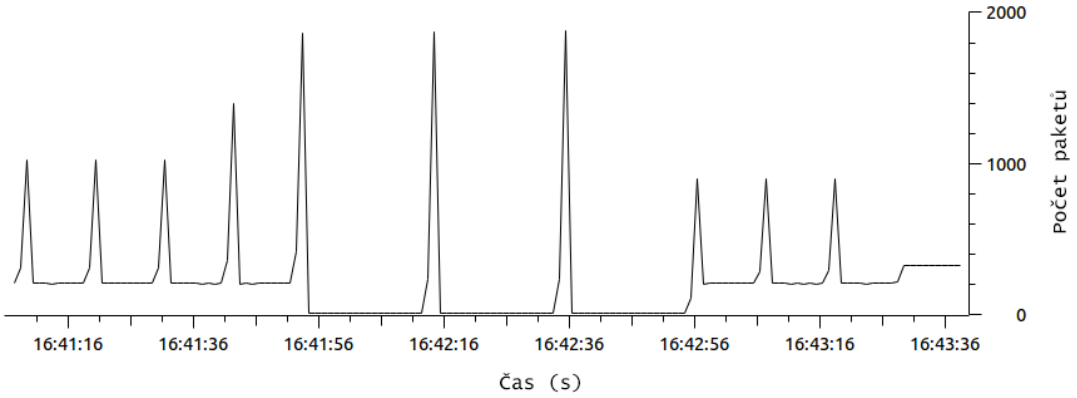
Do konzole je vypsáno:

```
Byl detekovan maximalni pocet 2000 UDP paketu za sekundu. Sit bude na 20 sekund
nedostupna - Podezreni na probihajici DDoS utok.
```

a programem následně do konzole zadaný příkaz pro zahození veškerého provozu z eth2.

```
ExecuteCommand ($"ebtables -I FORWARD -p IPv4 --ip-protocol udp -j DROP");
              var autoEvent = new AutoResetEvent (false);
              var timer = new Timer (MaximumTrafficUDPToShutDownPerSecond_Timer
                  , autoEvent,UDPMaximumTrafficToShutDownDuration * 1000,
                  Timeout.Infinite);
```

Výpis 5: C# .NET Core na Ubuntu 14.04. Ebtables podmínka pro zahození veškerého provozu na eth2 UDP provozu



Obrázek 38: UDP flood - Odchozí paketový tok na PC4 - eth0

Od začátku grafu [38] probíhala komunikace s redukcí, protože dosahovala toku přes 1000 paketů za sekundu. Provoz byl poté navýšen, což lze vidět na časové ose v 16:41:55 sekund a vidíme, že počet paketů překročil limit 2000 paketů za sekundu, provoz byl tedy na 20 sekund odstřihnut ze směru eth2 a program všechny pakety zahazoval. Můžeme vidět, že limit byl dosažen třikrát, pak se počet paketů snížil zpět k tisíci paketů za sekundu a pak už pokračoval normální neomezený provoz.

Po uplynutí zadaného časového parametru *UDPMaximumTrafficToShutDownPerSecond* byl programem vykonán příkaz pro smazání příkazu:

```
static void MaximumTrafficUDPToShutDownPerSecond_Timer (Object stateInfo)
      {
          completeShutDownUDP = false;
          ExecuteCommand ($"ebtables -D FORWARD -p IPv4 --ip-protocol udp -j
              DROP");
          Console.WriteLine ($"UDP provoz obnoven");
      }
```

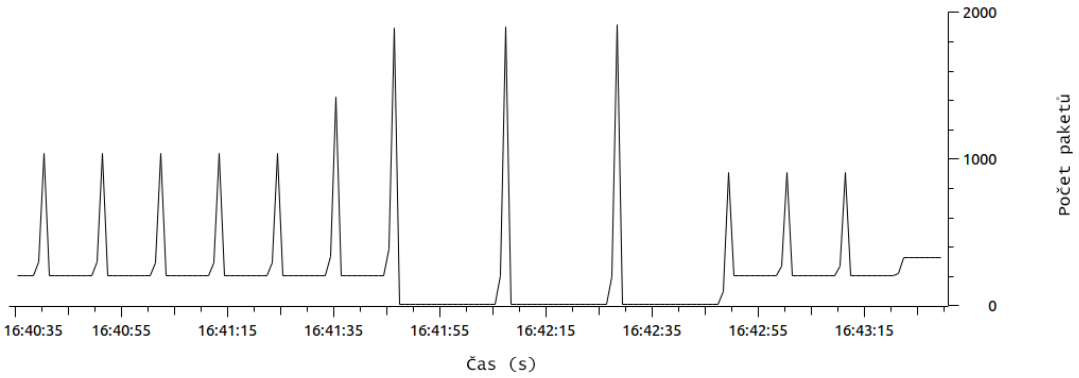
Výpis 6: C# .NET Core na Ubuntu 14.04. Příkaz pro smazání příkazu na zahazování UDP provozu

a byla vypsána hláška

```
UDP provoz obnoven.
```

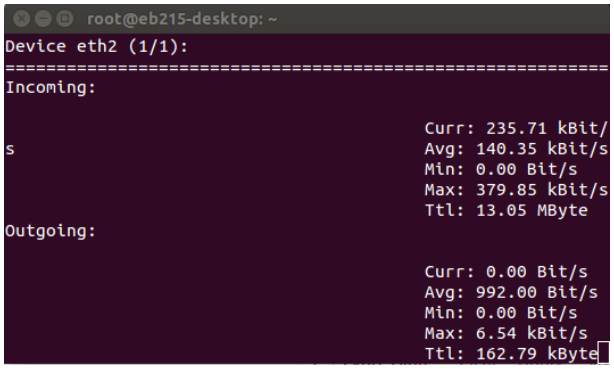
Avšak stejně jako u předešlého příkladu dosáhl limit maximálního počtu paketů za sekundu a síť byla opět periodicky na 20 sekund nedostupná, dokud se počet paketů nesnížil.

Z obrázku [38] pak vidíme, že paketový z eth2 na eth0 opravdu zablokován, po dosažení maximální limitu paketů na 20 sekund, což je ukázáno i v obrázku [39], kde vidíme, další průběh a potlačení paketového toku v delší časové ose.

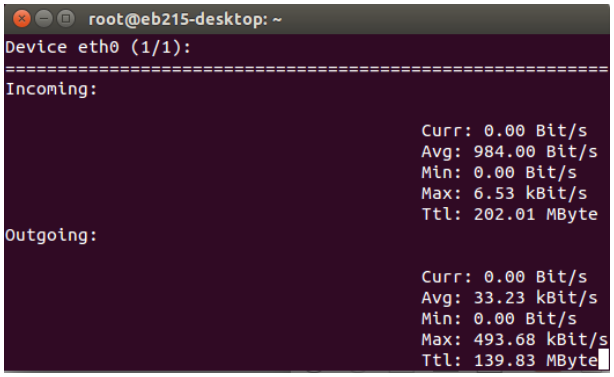


Obrázek 39: UDP flood: Příchozí paketový tok na Apache2 Server - eth0

Přitom na obrázku [40] můžeme vidět, že v důsledku zvětšení množství příchozích paketů se paketový tok zvýšil na 235 kBit/s. A na obrázku [41] můžeme vidět, že odchozí paketový tok byl opravdu nulový a dosahoval 0 Bit/s.

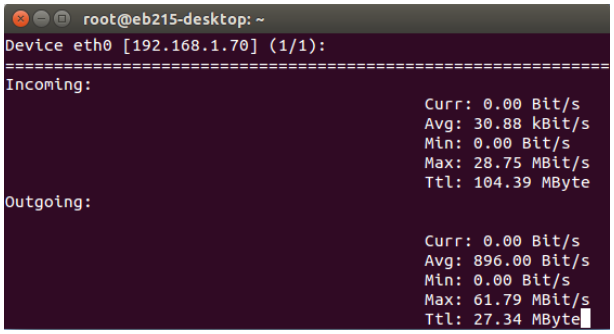


Obrázek 40: UDP flood: Příchozí paketový tok na PC4 - eth2



Obrázek 41: UDP flood: Odchozí paketový tok na PC4 - eth0

Na obrázku [42] pak můžeme vidět, že opravdu je příchozí paketový tok 0 Bit/s.



Obrázek 42: UDP flood: Příchozí paketový tok na Apache2 Server - eth0

U dalších simulací byla vynechána část se znázorněním programových částí, neboť se u podmínek měnil jen protokol a podmínky vypadají až na pár drobností úplně stejně. Ukázka funkce programu tedy byla znázorněna u UDP floodu. Celý program lze pak vidět v příloze, kde je přidán.

8.3.3 ICMP flood

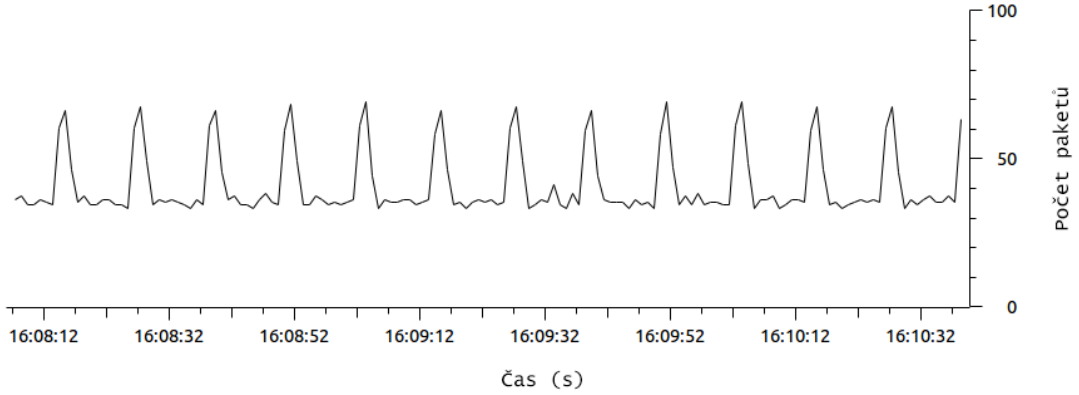
Simulace ICMP flood byla udělána už s méně podrobným počtem obrázků, než tomu bylo u UDP floodu. Funkce programu je stejná. Program reaguje na počet přijatých paketů za sekundu a reaguje podle našich zadaných parametrů. Pro ukázku jsem zvolil menší počet paketů na kterém půjde vidět pěkně práce programu.

Využit byl příkaz:

```
hping3 --fast -1 -i u70000 192.168.1.70
```

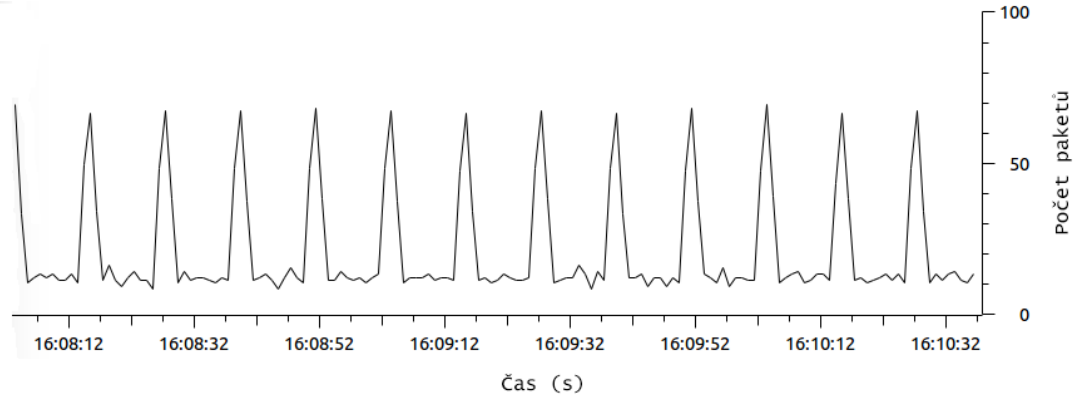
Zadané hodnoty byly:

```
Maximální počet paketů za sekundu, po které bude provoz omezen: 65
Maximální počet paketů na které bude provoz omezen: 5
Čas po kterou bude provoz omezen: 10
Maximální počet paketů po kterém bude síť nedostupná: 110
Čas po která bude síť nedostupná: 20
Maximální počet paketů za sekundu z jedné IP adresy: 80
Čas po který bude IP adresa bloknutá: 10
```

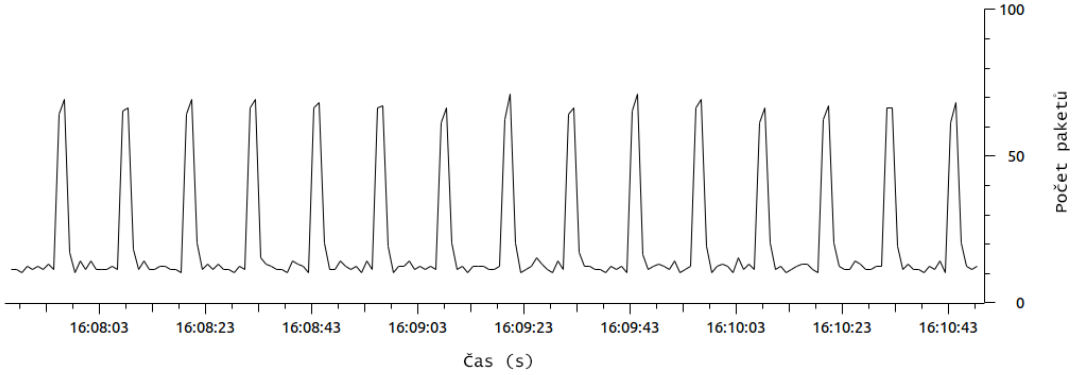


Obrázek 43: ICMP flood: Příchozí paketový tok na PC4 - eth2

Na obrázku [43] lze vidět, jak program při dosažení limitního počtu paketů za sekundu redukoval provoz. Na obrázku [44] pak můžeme vidět, že počet paketů dosahoval deseti. Mezi pakety na eth0 jsou i pakety *reply*, tedy musíme počítat, že počet paketů bude tedy dvojnásobný odpovídá-li napadené zařízení na *icmp requesty*.

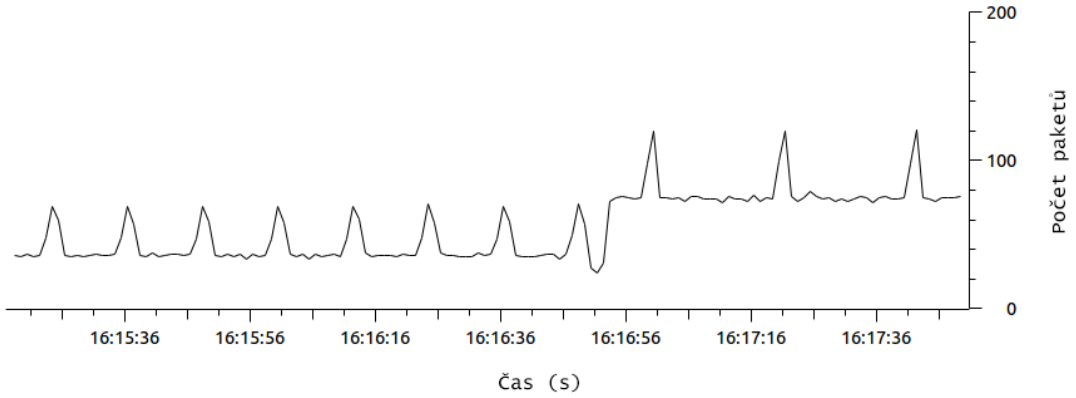


Obrázek 44: ICMP flood: Odchozí paketový tok na PC4 - eth0

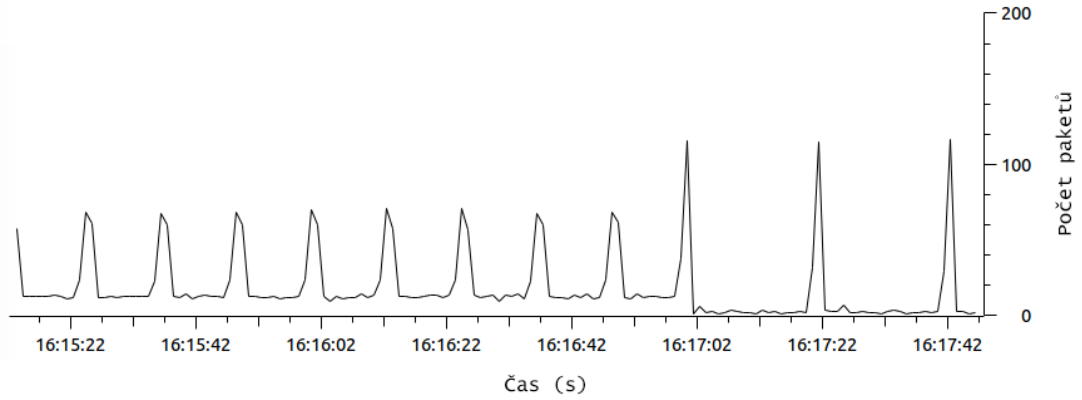


Obrázek 45: ICMP flood: Příchozí paketový tok na Apache2 Server - eth0

Překročí-li však počet paketů námi stanovenou mez 110 paketů za sekundu, program síť na dvacet sekund odstřihne, viz. obrázky [46], [47].

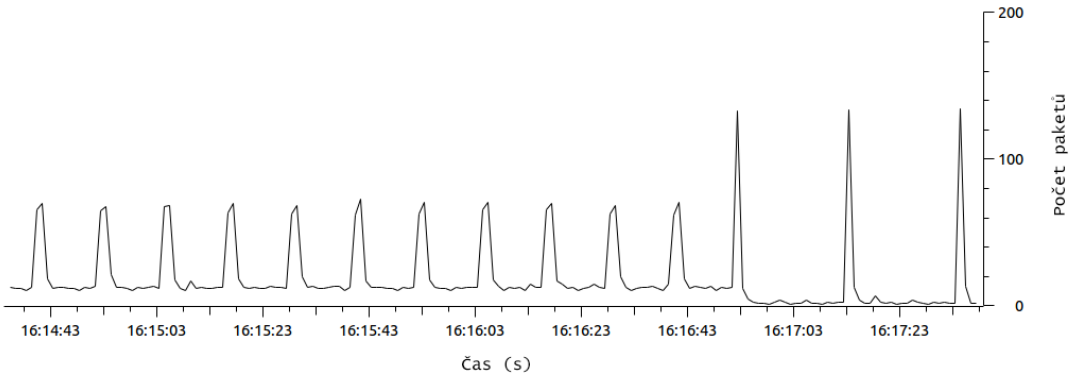


Obrázek 46: ICMP flood: Příchozí paketový tok na PC4 - eth2



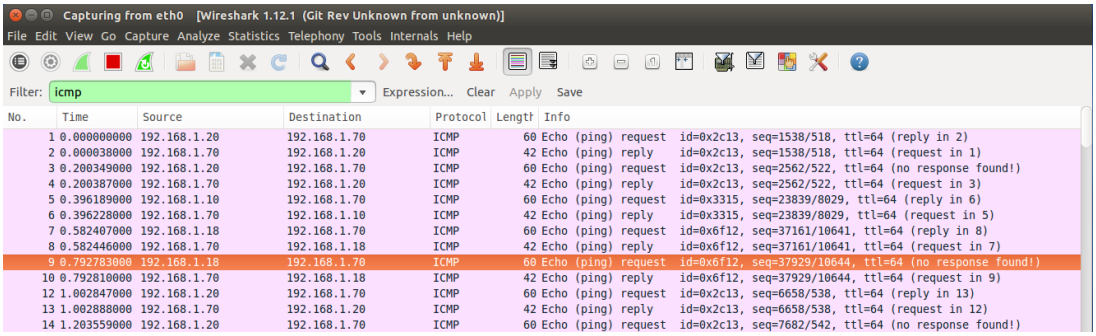
Obrázek 47: ICMP flood: Odchozím paketový tok na PC4 - eth0

Vídíme, že program při hraničním počtu paketů úspěšný zablokoval veškerý provoz na dvacet sekund na rozhraní eth2. Z obrázků lze také zpozorovat, že když bylo zpět posláno menší množství paketů, provoz byl úspěšně jen omezen na stanovenou mez.



Obrázek 48: ICMP flood: Příchozí paketový tok na Apache2 Server - eth0

Počet přijatých paketů na Apache2 Server na zadaný parametr 5 může být potvrzen vypsáním v paketovém analyzátoru Wireshark.



Obrázek 49: ICMP flood: Počet příchozích paketů na Apache2 Server po omezení programem

Vidíme, že počet přijatých paketů *ICMP request* se opravdu rovná přesně pěti za sekundu. Tím byla ověřena schopnost programu omezit, ale i zahazovat pakety při dosažení stanovených limitů u ICMP floodu.

8.3.4 TCP SYN flood

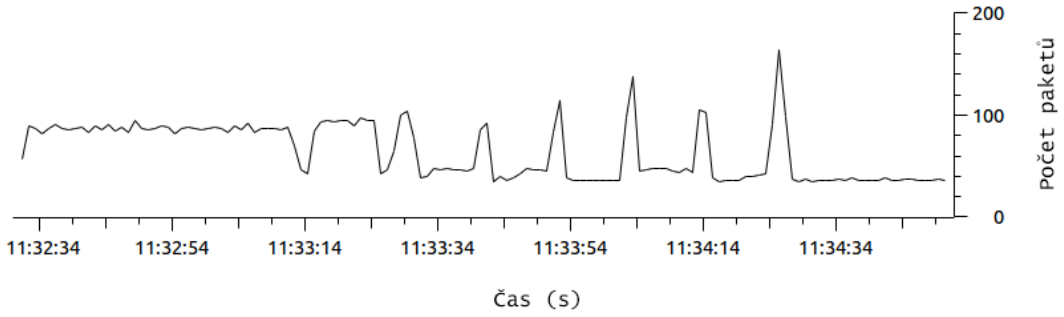
Simulace TCP flood byla udělána s těmito parametry:

Zadané hodnoty byly:

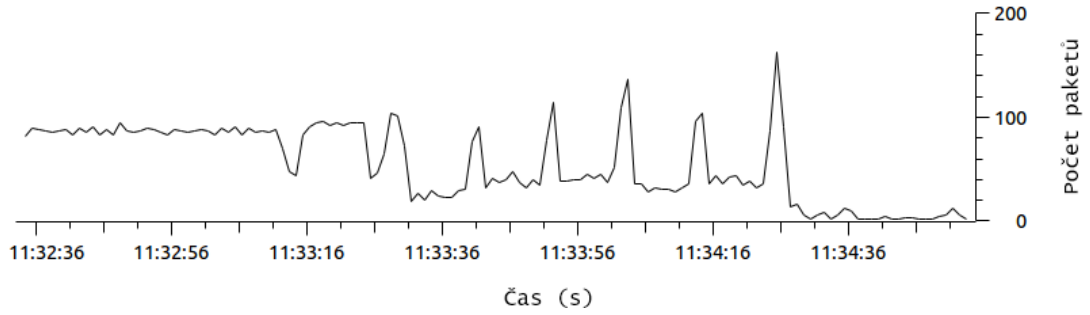
```
Maximální počet paketů za sekundu, po které bude provoz omezen: 100
Maximální počet paketů na které bude provoz omezen: 10
Čas po kterou bude provoz omezen: 10
Maximální počet paketů po kterém bude síť nedostupná: 170
Čas po která bude síť nedostupná: 30
Maximální počet paketů za sekundu z jedné IP adresy: 80
Čas po který bude IP adresa bloknutá: 10
```

a využit příkaz:

```
hping3 --fast -1 -i u40000 192.168.1.70
```



Obrázek 50: TCP SYN flood: Příchozí paketový tok na PC4 - eth2

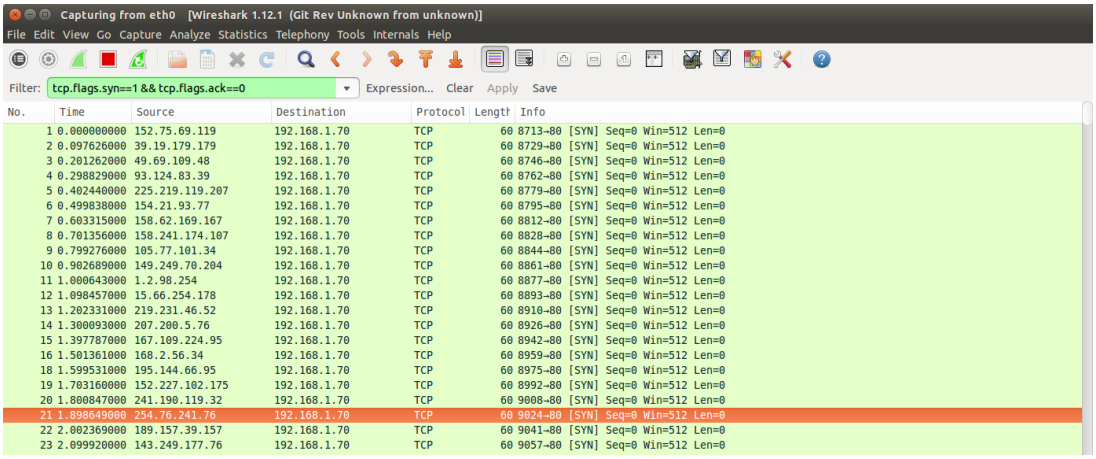


Obrázek 51: TCP SYN flood: Odchozí paketový tok na PC4 - eth0

Na obrázku[50] je možné vidět, jak paketový tok hraničně míjel hranici sto paketů za sekundu, to bylo simulováno proto, aby bylo vidět, že program opravdu reaguje až při dosažení stanovené hranice, v momentě, kdy ji překonal, bylo množství paketů za sekundu zredukováno na deset za sekundu. Avšak z odchozího provozu lze vidět, že počet paketů je na eth0 větší. Je to dáno vlastnostmi TCP protokolu.

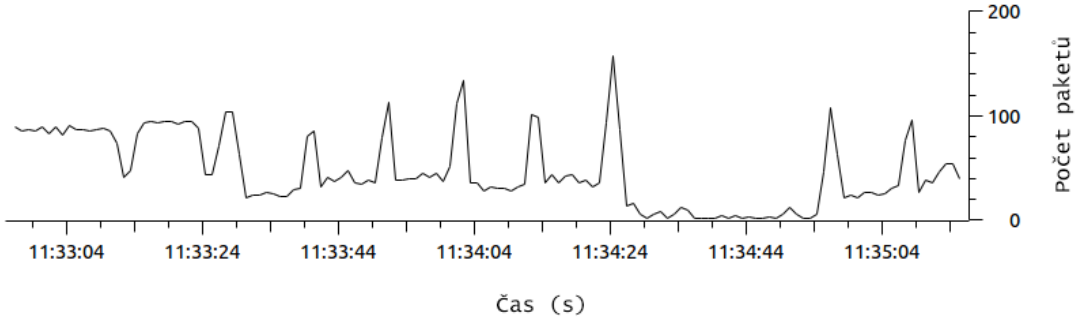
U TCP komunikace se snaží probíhat three-way handshake, tedy na SYN poslaný paket se počítač snaží reagovat SYN-ACK, proto je paketů na eth0 víc. Navíc bylo použito na některých útočících strojích při útoku příznaku *–rand-source* pro podvržení IP adres, takže také tak probíhá poslání pár RST paketů pro resetování spojení. Počet paketů omezených parametrem na 10 si můžeme ověřit z výpisu Wiresharku v [52].

Důvod, proč se podobá příchozí paketový tok odchozímu je ten, že na rozdíl od UDP protokolu, který jen posílá pakety k cíli, aniž by čekal na odpověď, protokol TCP má tzv. Sliding Window Protocol. To znamená, má-li příjemce problémy z nějakého důvodu odebírat včas data, například zahlcením linky a je velká ztrátovost paketů, odesílatel nedostává ACKy, snižuje se vysílací okno. Tím se sníží počet příchozích paketů na rozhraní eth2.



Obrázek 52: TCP SYN flood: Počet přijatých TCP SYN paketů na Apache2 Serveru - eth0

Z obrázku [52] je zřejmé, že počet paketů za sekundu je kolem desíti za sekundu, přesně jak jsme zadali v parametrech. Z dalšího obrázku [50] je vidět, že počet TCP paketů v čase přibližně 11:34:29, kdy byla komunikace po eth2 zablokována, stále posílala pakety, ale v menší míře kvůli velkému množství zahozených paketů. Odchozí provoz byl poté téměř nulový.



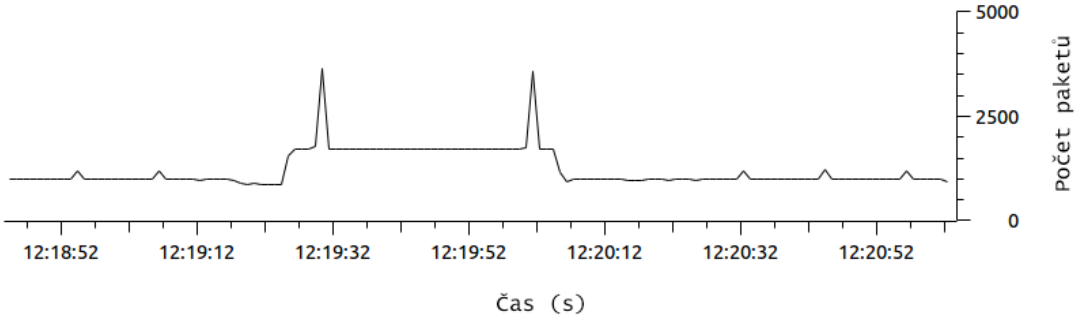
Obrázek 53: TCP SYN flood: Příchozí paketový tok na Apache2 Server - eth0

Na obrázku [53] vidíme, že odchozí pakety z eth0 PC4 je skoro totožné s přijatými pakety na Apache2 Server.

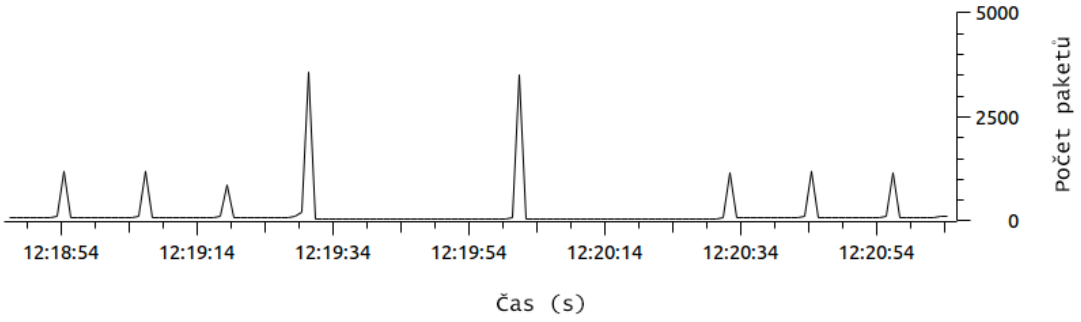
Program byl vyzkoušen s parametry:

```
Maximální počet paketů za sekundu, po které bude provoz omezen: 1000
Maximální počet paketů na které bude provoz omezen: 100
Čas po kterou bude provoz omezen: 10
Maximální počet paketů po kterém bude síť nedostupná: 3000
Čas po která bude síť nedostupná: 30
Maximální počet paketů za sekundu z jedné IP adresy: 2000
Čas po který bude IP adresa bloknutá: 10
```

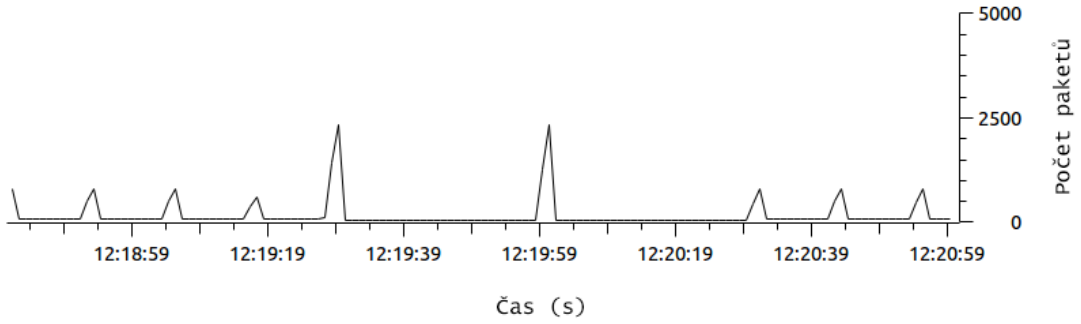
Byl vyzkoušen silnější paketový tok. Program pracoval pořád stejně i s chvilkovým nárůstem paketů nad tři tisíce paketů za sekundu, kdy úspěšně zablokoval na třicet sekund provoz jdoucí k Apache2 Server, jak můžeme vidět na obrázku [55] a [56].



Obrázek 54: TCP SYN flood: Příchozí paketový tok na PC4 - eth2



Obrázek 55: TCP SYN flood: Odchozí paketový tok na PC4 - eth0



Obrázek 56: TCP SYN flood: Příchozí paketový tok na Apache2 Server - eth0

Tímto jsme ověřili, že program zvládne i větší množství paketů, než bylo ukázáno u UDP, nebo ICMP.

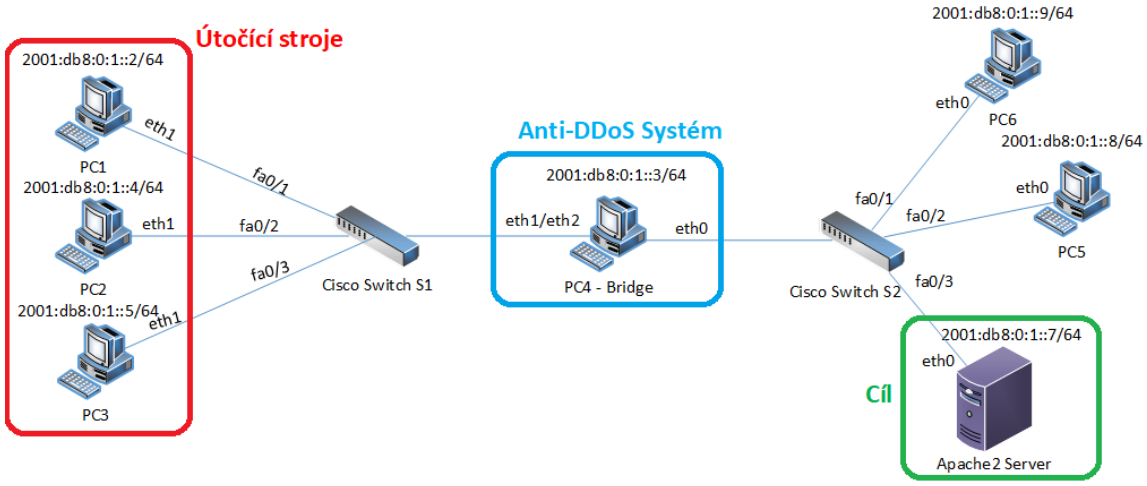
8.4 Testování programu v laboratorním prostředí s IPv6 adresováním

Pro budoucí použitelnost programu a ověření kompatibility s IPv6, byla vyzkoušena simulace s ICMPv6 pakety. Anti-DDoS program byl rozšířen o dalších 5 parametrů, které jsou určeny pro IPv6.

Pro generování provozu bylo nemožné použít generátor paketů hping3, protože nepodporuje IPv6. Použil jsem tedy pro generování paketů další ze zmíněných nástrojů pro útoky program Ostinato, který umožňuje generování i IPv6 paketů.

Následné zadávání množství paketů a protokolu se zadává v grafickém editoru programu Ostinato, kde je možné si zvolit protokol, verze paketu, cíl a množství paketů za sekundu.

8.4.1 Topologie



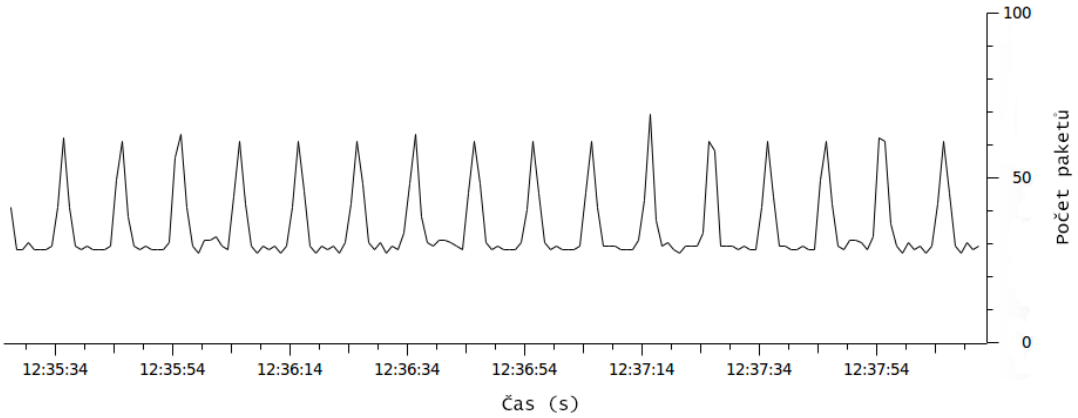
Obrázek 57: Topologie pro testování programu s IPv6 adresováním

Byla použita stejná topologie, jen s využitím IPv6 adres.

Nastavené hodnoty vypadaly takto:

```
Maximální počet paketů za sekundu, po které bude provoz omezen: 60
Maximální počet paketů na které bude provoz omezen: 5
Čas po kterou bude provoz omezen: 10
Maximální počet paketů po kterém bude síť nedostupná: 240
Čas po která bude síť nedostupná: 20
```

U času jsou použity parametry v sekundách. Po následném nasazení programu na provoz vypadal provoz následovně. Na obrázku [58] vidíme redukci paketů přicházejících k PC4 po eth2.



Obrázek 58: Příchozí provoz na PC4 s IPv6 - eth2

Po spuštění Anti-DDoS programu s parametry pro IPv6, program zachytil hraniční počet parametrů, která spadala do podmínky programu na omezení provozu:

```
if (currentTraffic > ICMPV6MaximumTrafficPerSecond && completeShutDownICMPV6 ==
    false) {
      var ruleName = _random.Next (0, 999999).ToString ();
      Console.WriteLine ($"Byl detekovan hranicni pocet {
          ICMPV6MaximumTrafficPerSecond} ICMPv6 paketu za sekundu. " +
      $"ICMP provoz bude omezen na {ICMPV6MaximumTrafficPacketPerSecond}
          paketu za sekundu na dobu {ICMPV6MaximumTrafficReductionDuration}
          sekund.");
```

Výpis 7: C# .NET Core na Ubuntu 14.04. Podmínka na omezení ICMPv6 provozu

Po splnění této podmínky se přímo vykoná příkaz s ebtables na redukci provozu dle parametru *ICMPV6MaximumTrafficPacketPerSecond* zadaného na začátku programu.

Do konzole bylo vypsáno:

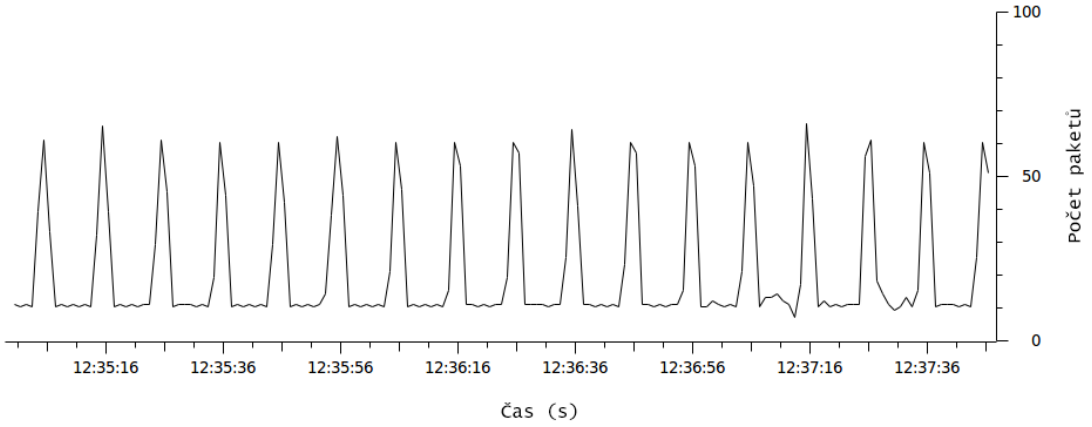
```
Byl detekovan hranicni pocet 60 ICMPv6 paketu za sekundu.
ICMPv6 provoz bude omezen na 5 paketu za sekundu na dobu 10 sekund.
```

```
  ExecuteCommand($"ebtables -N {ruleName};" +
                 $"ebtables -A FORWARD -p IPv6 --ip6-protocol ipv6-icmp -j {
                     ruleName};"+
                 $"ebtables -A {ruleName} -p IPv6 --limit {
                     ICMPV6MaximumTrafficPacketPerSecond}/second --limit-burst 3
                     -j ACCEPT;" +
                 $"ebtables -A {ruleName} -j DROP");
```

```
            var timer = new Timer (MaximumTrafficICMPV6PerSecond_Timer,
                ruleName, ICMPV6MaximumTrafficReductionDuration * 1000,
                Timeout.Infinite);
        }
}
```

Výpis 8: C# .NET Core na Ubuntu 14.04. Ebtables příkazy v programu na omezení UDP provozu

Po splnění podmínek na omezení ICMPv6 provozu, se tok paketů z PC4 zredukuje na zadané množství *ICMPV6MaximumTrafficPacketPerSecond*, jak můžeme vidět z odchozího paketového grafu [59].



Obrázek 59: Odchozí paketový tok z PC4 s IPv6 - eth2

Jak lze z obrázku [59] vidět, opět je na rozhraní okolo deseti paketů za sekundu, kvůli Echo Reply paketům.

Po uplynutí desíti sekund byl příkaz programem zrušen.

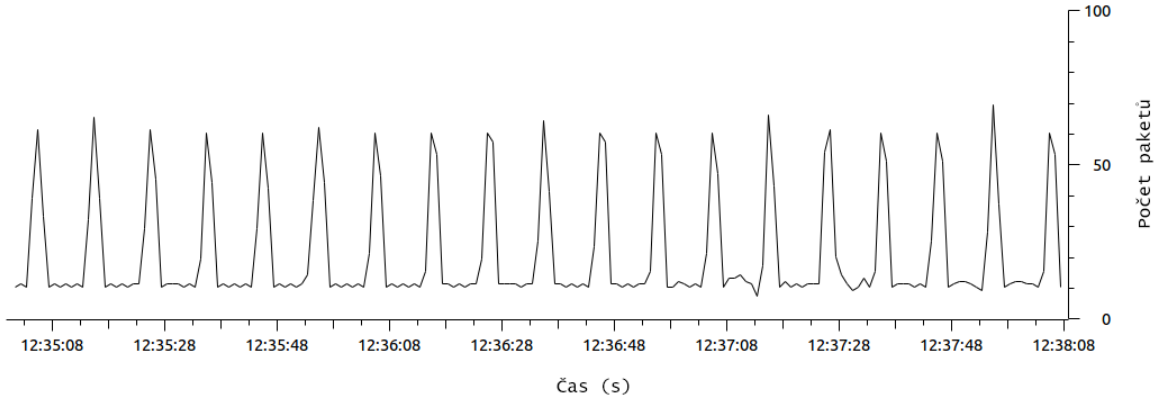
```
static void MaximumTrafficICMPV6PerSecond_Timer (Object stateInfo){
        var ruleName = stateInfo.ToString ();
        ExecuteCommand ($"ebtables -D FORWARD -p IPv6 --ip6-protocol ipv6-
            icmp -j {ruleName};" +
                        $"ebtables -D {ruleName} -p IPv6 --limit {
                            ICMPV6MaximumTrafficPacketPerSecond}/second --
                            limit-burst 3 -j ACCEPT;" +
                    $"ebtables -D {ruleName} -j DROP;" +
                    $"ebtables -X {ruleName}");
}
```

Výpis 9: C# .NET Core na Ubuntu 14.04. Ebtables příkazy v programu na omezení UDP provozu

a do konzole je vypsáno:

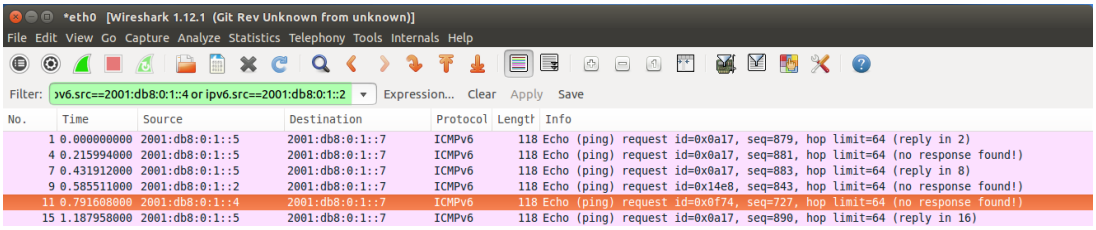
`ICMPv6 provoz po omezení obnoven.`

Tok paketů přijatých na Apache2 Serveru poté vypadal následovně.



Obrázek 60: Příchozí paketový tok na Apache2 Server s IPv6 - eth0

Omezení tedy odpovídalo počtu 5 paketů za sekundu a provoz byl redukován co deset sekund. Počet 5 paketů ICMPv6 Echo Request za sekundu potvrzuje i výpis z Wiresharku z Apache2 Server[**??**].



Obrázek 61: ICMPv6 flood: Příchozí paketový tok na Apache2 Server

Překročíme-li opět hranici zadaného parametru *ICMPV6MaximumTrafficToShutDownPerSecond*, provoz bude podle zadaného parametru *ICMPV6MaximumTrafficToShutDownDuration* 20 sekund zahazován.

Splní se podmínka programu pro zahazování provozu pro ICMPv6.

```
if (currentTraffic > ICMPV6MaximumTrafficToShutDownPerSecond) {
    Console.WriteLine ($"Byl detekovan maximalni pocet {
        ICMPV6MaximumTrafficToShutDownPerSecond} ICMPv6 paketu za sekundu. " +
    $"Sit bude na {ICMPV6MaximumTrafficToShutDownDuration} sekund nedostupna -
        Podezreni na probihajici DDoS utok");
    completeShutDownICMP = true;
```

Výpis 10: C# .NET Core na Ubuntu 14.04. Ebtables příkazy v programu na omezení UDP provozu

A vykoná se příkaz pro zahazování provozu.

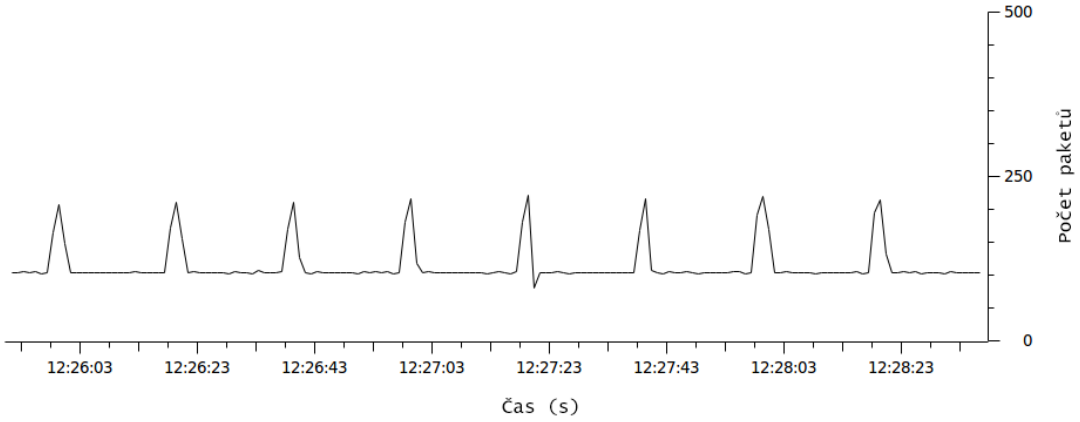
```
ExecuteCommand($"ebtables -I FORWARD -p IPv6 --ip6-protocol ipv6-icmp -j DROP"
    );
            var autoEvent = new AutoResetEvent (false);
            var timer = new Timer (
                MaximumTrafficICMPV6ToShutDownPerSecond_Timer,
            autoEvent, ICMPV6MaximumTrafficToShutDownDuration * 1000, Timeout
                .Infinite);
}
```

Výpis 11: C# .NET Core na Ubuntu 14.04. Ebtables příkazy v programu na omezení UDP provozu

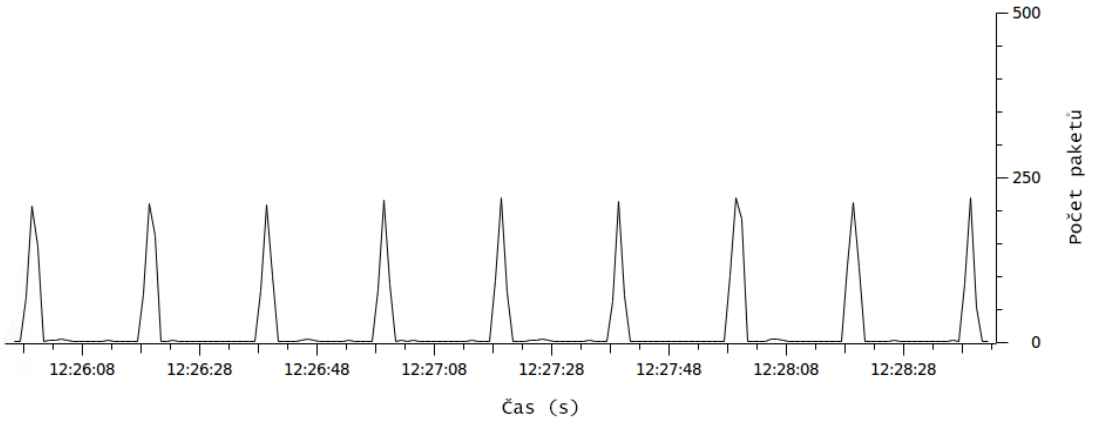
Do konzole bylo vypsáno:

```
Byl detekovan maximalni pocet 240 ICMPv6 paketu za sekundu.
Sit bude na 20 sekund nedostupna - Podezreni na probihajici DDoS utok
```

Příchozí a odchozí provoz z PC4 poté vypadá následovně



Obrázek 62: Příchozí paketový tok na PC4 s IPv6 - eth2



Obrázek 63: Odchozí paketový tok z PC4 s IPv6 - eth0

Periodicky se co 20 sekund celá operace opakovala s využitím příkazu pro smazání příkazu:

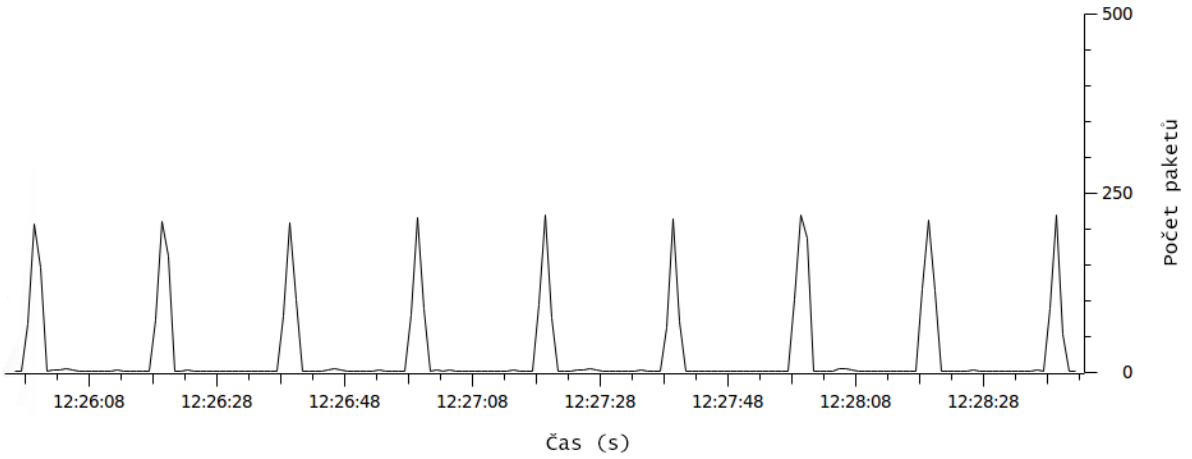
```
static void MaximumTrafficICMPV6ToShutDownPerSecond_Timer (Object stateInfo)
            {
            completeShutDownICMPV6 = false;
            Console.WriteLine ($"ICMPv6 provoz obnoven.");
            ExecuteCommand ($"ebtables -D FORWARD -p IPv6 --ip6-protocol icmp -
                j DROP");
}
```

Výpis 12: C# .NET Core na Ubuntu 14.04. Ebtables příkazy v programu na omezení UDP provozu

a do konzole je vypsáno

```
ICMPv6 provoz obnoven.
```

Počet paketů na Apache2 Serveru poté vypadá stejně jako výpis z odchozího provozu z PC4.



Obrázek 64: Příchozí paketový tok na Apache2 Server s IPv6 - eth0

Přesně dle zadaného parametru *ICMPV6MaximumTrafficToShutDownDuration*, se celá operace periodicky opakuje.

9 Závěr

Jedním z cílů práce bylo seznámit čtenáře s problematikou DoS/DDoS útoků, které se postupně stávají nejčastější formou útoku vůbec, základními vlastnostmi, rozdělením a úzce spjatým tématem botnet sítěmi, které se používají k realizaci útoku. Problémem dneška se stává velký rozmach IoT zařízení. Díky nim dnes čelíme útokům, které se svou silou dostávají na úplně novou úroveň. V sekci o aktuální situaci byl vykreslen graf, který ukazuje nepopiratelnou rostoucí sílu těchto útoků.

Po popsání těchto základních problematik a seznámení s tématem byly teoreticky popsány typy síťových počítačových útoků a metody obrany před nimi. Popsány byly také nejčastěji používané nástroje pro útok jako HOIC a LOIC, zmíněné byly nástroje, použité v praktické části. Po seznámení čtenáře s nutnou teoretickou části, jsem přistoupil k realizaci praktické části.

Cílem praktické části diplomové práce bylo navržení otevřeného systému pro zachycení a blokování útoků využívajících odepření služeb. K dosažení tohoto cíle byl vytvořen program napsán v C# .NET Coru, který byl spuštěn na Ubuntu 14.04 pomocí MonoDevelop. Pro programování programu v C# .NET Coru jsem se rozhodl kvůli možnosti spustit program bez větších problémů na jiných operačních systémech a plnil tak v maximální míře open-source projekt. Tento program byl spuštěn na počítači, který odděloval dvě sítě a tvořil jakýsi filtr mezi nimi. K filtrování provozu byly použity příkazy ebtables, které jsem díky programu dokázal efektivně zautomatizovat. Do programu byly zadávány hodnoty, které udávaly počet paketů na omezení, či úplné potlačení útoku. Program dokázal při dosažení těchto hodnot efektivně omezit, či úplně zamezit útokům na námi stanovený čas, který jsme taktéž zadávali se spuštěním programu.

V sekci simulace útoku na koncové zařízení bez ochrany jsem vyzkoušel možnosti používaného programu Hping3 a počítačů, na kterých jsem ve škole pracoval. Při útocích s maximálním počtem vyslaných paketů, počítače nebyly schopny pracovat s velkým objemem paketů. Pro tento fakt byla zvolena práce s menším počet vyslaných paketů k Apache2 Serveru, která slouží k ukázce funkčnosti programu. Jako výstup je potom ukázán rozdíl mezi rozhraním příchozích a odchozích paketů zachycených v paketovém analyzátoru a dostupnosti Apache2 Serveru z počítačů za Anti-DDoS systémem, společně s příchozím paketovým tokem vykresleným paketovým analyzátorem na rozhraním jdoucím k Apache2 Serveru.

Simulován byl útok ICMP flood, UDP flood a TCP SYN flood. Pro simulování byl použit jeden z programů zmíněných v části, která se zaměřovala nástroji pro útoky. Program Hping3 byl otestován v laboratoři a dokázal být dostatečně flexibilním nástrojem pro simulování variabilního počtu paketů. Při spuštění programu bylo zadáváno po sedmi hodnotách pro každý protokol, tedy celkem 21 parametrů. Tento počet zaručuje uživateli maximální možnou flexibilitu a přizpůsobení programu infrastrukturním podmínkám, kde je program použit.

Vytvořený program dokázal úspěšně po zadaných hraničních hodnotách omezit tok paketů na cíl, nebo při dosažení maximálního zadaného parametru útok úplně potlačit u všech třech

zmíněných útoků tok paketů k cíli na zadaný časový úsek. Do programu také byla přidána možnost, kdy můžeme zadat maximální počet paketů z jedné IP adresy, který je nadměrný. Výhodou pak je, že se zakáže na náš daný časový úsek jen IP adresa, která tuto hranici překročí, ostatní provoz zůstává neomezen, pokud nepřekročí jiný ze zadaných parametrů.

Pro budoucí využití programu jsem v poslední části simulace využil možnost otestování programu na IPv6 adresách. Do programu bylo úspěšně zakomponováno dalších 5 parametrů pro IPv6, pro omezení či celkového potlačení útoku u ICMPv6 flood. Potlačování provozu probíhalo po malých úpravách taktéž pomocí ebtables. Výhodou a správnou volbou pro napsání programu v C# .NET Coru se ukázala hned při této volbě na částečné rozšíření o IPv6. Knihovny, které jsou používány programem jako PacketDotNet a další, už počítají s rozšířením na IPv6 a nebylo tak nutné velkých kódových úprav. Pro generování útoku s IPv6 pakety byl použit další program pro generování útoku zmíněných v teoretické části. Program Ostinato dokázal na rozdíl od programu Hping3 pracovat s IPv6 adresami.

V práci jsou uvedeny také zdrojové ukázky programu. Celý vytvořený Anti DDoS program je uveden v příloze.

Bc. Petr Müller

Literatura

[1] Bhattacharyya D.K., Kalita J.K. *DDoS Attacks*: Evolution, Detection, Prevention, Reaction, and Tolerance. CRC Press 2016

[2] Mirkovic J. *Internet Denial of Service*: Attack and Defense Mechanisms. Prentice Hall 2005

[3] DDoS Attack Report 2017 Dostupné z: [online]. Dostupné z: http://www.aiuken.com/uae/news/ddos-attacks-report-2017-18.html

[4] 602 Gbps! This May Have Been the Largest DDoS Attack in History. The Hacker News — Online Cyber Security News & Analysis [online]. Dostupné z: https://thehackernews.com/2016/01/biggest-ddos-attack.html

[5] Pronajmu botnet, ceník přiložen... aneb jak se dělá DDoS - Root.cz. Root.cz - informace nejen ze světa Linuxu [online]. Copyright 1998 [cit. 05.03.2018]. Dostupné z: https://www.root.cz/clanky/pronajmu-botnet-cenik-prilozen-aneb-jak-se-dela-ddos/

[6] Intelligent DDoS Mitigation System-IDMS. Cisco Networking Tutorials [online]. Dostupné z: https://www.networkstraining.com/intelligent-ddos-mitigation-system-idms/

[7] A 1.3-Tbs DDoS Hit GitHub, the Largest Yet Recorded | WIRED. WIRED [online]. Dostupné z: https://www.wired.com/story/github-ddos-memcached/

[8] 1,7 Tbps DDoS Attack - Netscout Arbor[online]. Dostupné z: https://www.arbornetworks.com/blog/asert/netscout-arbor-confirms-1-7-tbps-ddos-attack-terabit-attack-era-upon-us/

[9] Secure communication using remote procedure calls: Andrew D. Birrell , Bruce Jay Nelson, Implementing remote procedure calls, ACM Transactions on Computer Systems (TOCS), v.2 n.1, p.39-59, February 1984 [doi>10.1145/2080.357392]

[10] DDoS Survival Handbook [online]. Dostupné z: https://www.security.radware.com/ uploadedFiles/Resources _and/_Content/DDoS_Handbook/DDoS_Handbook.pdf

[11] Types of DDoS attacks | FlowGuard. Effective protection against DDoS attacks | FlowGuard [online]. Copyright ComSource s.r.o. [cit. 17.04.2018]. Dostupné z: https://www.flowguard.io/about-ddos/types-of-ddos/

[12] MonoDevelop. MonoDevelop | MonoDevelop [online]. Copyright 2018 MonoDevelop Project [cit. 17.04.2018]. Dostupné z: https://www.monodevelop.com/download/#fndtn-download-lin

[13] .NET and C# - Get Started in 10 Minutes. Microsoft Corporation [online]. Dostupné z: https://www.microsoft.com/net/learn/get-started/linux/ubuntu14-04

[14] GitHub - chmorgan/sharppcap: Official repository - Fully managed, cross platform (Windows, Mac, Linux) .NET library for capturing packets. The world's leading software development platform · GitHub [online]. Copyright 2018 [cit. 17.04.2018]. Dostupné z: https://github.com/chmorgan/sharppcap

[15] Types of DDoS Attacks. eSecurity Planet: Internet Security for IT Professionals [online]. Dostupné z: https://www.esecurityplanet.com/network-security/types-of-ddos-attacks.html

[16] Ostinato Network Traffic Generator. Ostinato Network Traffic Generator [online]. Copyright 2018 Srivats P. Powered by [cit. 17.04.2018]. Dostupné z: https://ostinato.org/

[17] Hping - Active Network Security Tool. Hping - Active Network Security Tool [online]. Dostupné z: http://www.hping.org/

[18] GoldenEye | wroot. wroot [online]. Copyright wroot [cit. 17.04.2018]. Dostupné z: https://wroot.org/projects/goldeneye/

[19] DDoS Mitigation & Network Visibility Solutions | Arbor Networks [online].Copyright[cit. 17.04.2018].Dostupné z: https://www.arbornetworks.com/images/documents/WISR2016_EN_Web.pdf

A Zdrojový kód

```
using System;
using System.Collections.Generic;
using SharpPcap;
using PacketDotNet;
using System.Diagnostics;
using System.Threading;

namespace DDos
{
    class Program
    {
        static Dictionary<string, int> TCPAdressesCount;
        static int TCPPacketCount;

        static Dictionary<string, int> UDPAdressesCount;
        static int UDPPacketCount;

        static Dictionary<string, int> ICMPAdressesCount;
        static int ICMPPacketCount;

        static Dictionary<string, int> ICMPV6AdressesCount;
        static int ICMPV6PacketCount;

        static Random _random;

        static bool completeShutDownTCP, completeShutDownUDP,
            completeShutDownICMP, completeShutDownICMPV6;

        static int TCPMaximumTrafficPerSecond;
        static int TCPMaximumTrafficPacketPerSecond;
        static int TCPMaximumTrafficReductionDuration;

        static int TCPMaximumTrafficToShutDownPerSecond;
        static int TCPMaximumTrafficToShutDownDuration;

        static int TCPMaximumTrafficPerIPToShutDownPerSecond;
```

```
static int TCPMaximumTrafficPerIPToShutDownDuration;

static int UDPMaximumTrafficPerSecond;
static int UDPMaximumTrafficPacketPerSecond;
static int UDPMaximumTrafficReductionDuration;

static int UDPMaximumTrafficToShutDownPerSecond;
static int UDPMaximumTrafficToShutDownDuration;

static int UDPMaximumTrafficPerIPToShutDownPerSecond;
static int UDPMaximumTrafficPerIPToShutDownDuration;

static int ICMPMaximumTrafficPerSecond;
static int ICMPMaximumTrafficPacketPerSecond;
static int ICMPMaximumTrafficReductionDuration;

static int ICMPMaximumTrafficToShutDownPerSecond;
static int ICMPMaximumTrafficToShutDownDuration;

static int ICMPMaximumTrafficPerIPToShutDownPerSecond;
static int ICMPMaximumTrafficPerIPToShutDownDuration;

static int ICMPV6MaximumTrafficPerSecond;
static int ICMPV6MaximumTrafficPacketPerSecond;
static int ICMPV6MaximumTrafficReductionDuration;

static int ICMPV6MaximumTrafficToShutDownPerSecond;
static int ICMPV6MaximumTrafficToShutDownDuration;

static void Main (string [] args)
{
    TCPAdressesCount = new Dictionary<string, int> ();
    TCPPacketCount = 0;

    UDPAdressesCount = new Dictionary<string, int> ();
    UDPPacketCount = 0;

    ICMPAdressesCount = new Dictionary<string, int> ();
    ICMPPacketCount = 0;
```

```
    ICMPV6AdressesCount = new Dictionary<string, int> ();
    ICMPV6PacketCount = 0;
    _random = new Random ();

    completeShutDownUDP = false;
    completeShutDownTCP = false;
    completeShutDownICMP = false;
    completeShutDownICMPV6 = false;

    ExecuteCommand ($"ebtables -F; ebtables -X");

    Init (args);

    CaptureDeviceList devices = CaptureDeviceList.Instance;

    foreach (var dev in devices)
        Console.WriteLine ($"{dev.Name}");

    var device = devices [0];
    device.Open (DeviceMode.Promiscuous);

    string filter = "tcp or udp or icmp";
    device.Filter = filter;

    device.OnPacketArrival += new PacketArrivalEventHandler (
        device_OnPacketArrival);
    device.StartCapture ();

    var autoEvent = new AutoResetEvent (false);
    new Timer (TCPTimer_Timer, autoEvent, 10, 1000);
    new Timer (UDPTimer_Timer, autoEvent, 35, 1000);
    new Timer (ICMPTimer_Timer, autoEvent, 71, 1000);
    new Timer (ICMPV6Timer_Timer, autoEvent, 81, 1000);
}

private static void device_OnPacketArrival (object sender,
    CaptureEventArgs e)
{
```

```
var packet = Packet.ParsePacket (e.Packet.LinkLayerType, e.Packet.
    Data);
var type = (PacketDotNet.IpPacket)packet.PayloadPacket;

if (type.Protocol == IPProtocolType.UDP) {
    UDPPacketCount++;

    var ipPacket = (PacketDotNet.IpPacket)packet.PayloadPacket;
    System.Net.IPAddress srcIp = ipPacket.SourceAddress;

    if (UDPAdressesCount.ContainsKey (srcIp.ToString ()))
        UDPAdressesCount [srcIp.ToString ()] = UDPAdressesCount [
            srcIp.ToString ()] + 1;
    else UDPAdressesCount.Add (srcIp.ToString (), 1);
}

if (type.Protocol == IPProtocolType.ICMP) {
    ICMPPacketCount++;

    var ipPacket = (PacketDotNet.IpPacket)packet.PayloadPacket;
    System.Net.IPAddress srcIp = ipPacket.SourceAddress;

    if (ICMPAdressesCount.ContainsKey (srcIp.ToString ()))
        ICMPAdressesCount [srcIp.ToString ()] = ICMPAdressesCount [
            srcIp.ToString ()] + 1;
    else ICMPAdressesCount.Add (srcIp.ToString (), 1);
}

if (type.Protocol == IPProtocolType.ICMPV6) {
    ICMPV6PacketCount++;

    var ipv6Packet = (PacketDotNet.IPv6Packet)packet.PayloadPacket;
    System.Net.IPAddress srcIp = ipv6Packet.SourceAddress;

    if (ICMPV6AdressesCount.ContainsKey (srcIp.ToString ()))
        ICMPV6AdressesCount [srcIp.ToString ()] = ICMPV6AdressesCount
            [srcIp.ToString ()] + 1;
    else ICMPV6AdressesCount.Add (srcIp.ToString (), 1);
}
```

```
    if (type.Protocol == IPProtocolType.TCP) {
        TCPPacketCount++;

        var ipPacket = (PacketDotNet.IpPacket)packet.PayloadPacket;
        System.Net.IPAddress srcIp = ipPacket.SourceAddress;

        if (TCPAdressesCount.ContainsKey (srcIp.ToString ()))
            TCPAdressesCount [srcIp.ToString ()] = TCPAdressesCount [
                srcIp.ToString ()] + 1;
        else TCPAdressesCount.Add (srcIp.ToString (), 1);
    }
}

static void TCPTimer_Timer (Object stateInfo)
{
    var currentTraffic = TCPPacketCount;

    Console.WriteLine ($"TCPPacketCount/s: {currentTraffic}");

    if (currentTraffic > TCPMaximumTrafficToShutDownPerSecond) {
        Console.WriteLine ($"Byl detekovan maximalni pocet {
            TCPMaximumTrafficToShutDownPerSecond} TCP za sekundu. " +
                        $"Sit bude na {
                            TCPMaximumTrafficToShutDownDuration} sekund
                            nedostupna - Podezreni na probihajici DDoS
                            utok");
        completeShutDownTCP = true;
        ExecuteCommand($"ebtables -I FORWARD -p IPv4 --ip-protocol tcp -
            j DROP");
        var autoEvent = new AutoResetEvent (false);
        var timer = new Timer (
            MaximumTrafficTCPToShutDownPerSecond_Timer, autoEvent,
            TCPMaximumTrafficPerIPToShutDownDuration * 1000, Timeout.
            Infinite);
    } else if (currentTraffic > TCPMaximumTrafficPerSecond &&
        completeShutDownTCP == false) {
        var ruleName = _random.Next (0, 999999).ToString ();
```

```
        Console.WriteLine ($"Byl detekovan hranicni pocet {
            TCPMaximumTrafficPerSecond} TCP paketu za sekundu. " +
                        $"TCP provoz bude omezen na {
                            TCPMaximumTrafficPacketPerSecond} paketu za
                            sekundu na dobu {
                            TCPMaximumTrafficReductionDuration} sekund."
                            );
        ExecuteCommand($"ebtables -N {ruleName};" +
                       $"ebtables -A FORWARD -p IPv4 --ip-protocol tcp -j
                            {ruleName};" +
                       $"ebtables -A {ruleName} -p IPv4 --limit {
                           TCPMaximumTrafficPacketPerSecond}/second --
                           limit-burst 3 -j ACCEPT;" +
                       $"ebtables -A {ruleName} -j DROP");
        var timer = new Timer (MaximumTrafficTCPPerSecond_Timer,
            ruleName, TCPMaximumTrafficReductionDuration * 1000, Timeout.
            Infinite);
    } else {
        foreach (var ip in TCPAdressesCount.Keys) {
            if (TCPAdressesCount [ip] >
                TCPMaximumTrafficPerIPToShutDownPerSecond) {
                Console.WriteLine ($"Byl detekovan hranicni pocet {
                    TCPMaximumTrafficPerIPToShutDownPerSecond} TCP paketu
                    z jedne adresy za sekundu." +
                                $"Adresa { ip.ToString ()} zablokovana
                                    na {
                                    TCPMaximumTrafficPerIPToShutDownDuration
                                    } sekund.");
                ExecuteCommand($"ebtables -A FORWARD -p IPv4 --ip-source
                    {ip.ToString ()} -j DROP");
                var autoEvent = new AutoResetEvent (false);
                var timer = new Timer (
                    MaximumTrafficPerIPToShutDownPerSecond_Timer, ip.
                    ToString (), TCPMaximumTrafficPerIPToShutDownDuration
                    * 1000, Timeout.Infinite);
            }
        }
    }
```

```
    TCPPacketCount = 0;
    TCPAdressesCount.Clear ();
}

static void UDPTimer_Timer (Object stateInfo)
{
    var currentTraffic = UDPPacketCount;

    Console.WriteLine ($"UDPPacketCount/s: {currentTraffic}");

    if (currentTraffic > UDPMaximumTrafficToShutDownPerSecond) {
        Console.WriteLine ($"Byl detekovan maximalni pocet {
            UDPMaximumTrafficToShutDownPerSecond} UDP paketu za sekundu.
            " +
                        $"Sit bude na {
                            UDPMaximumTrafficToShutDownDuration} sekund
                            nedostupna - Podezreni na probihajici DDoS
                            utok");
        completeShutDownUDP = true;
        ExecuteCommand($"ebtables -I FORWARD -p IPv4 --ip-protocol udp -
            j DROP");
        var autoEvent = new AutoResetEvent (false);
        var timer = new Timer (
            MaximumTrafficUDPToShutDownPerSecond_Timer, autoEvent,
            UDPMaximumTrafficToShutDownDuration * 1000, Timeout.Infinite)
            ;
    } else if (currentTraffic > UDPMaximumTrafficPerSecond &&
        completeShutDownTCP == false) {
        var ruleName = _random.Next (0, 999999).ToString ();
        Console.WriteLine ($"Byl detekovan hranicni pocet {
            UDPMaximumTrafficPerSecond} UDP paketu za sekundu. " +
                        $"TCP provoz bude omezen na {
                            UDPMaximumTrafficPacketPerSecond} paketu za
                            sekundu na dobu {
                            UDPMaximumTrafficReductionDuration} sekund."
                            );
        ExecuteCommand($"ebtables -N {ruleName};" +
                       $"ebtables -A FORWARD -p IPv4 --ip-protocol tcp -j
                           {ruleName};" +
```

```
                        $"ebtables -A {ruleName} -p IPv4 --limit {
                            UDPMaximumTrafficPacketPerSecond}/second --
                            limit-burst 3 -j ACCEPT;" +
                        $"ebtables -A {ruleName} -j DROP");
            var timer = new Timer (MaximumTrafficUDPPerSecond_Timer,
                ruleName, UDPMaximumTrafficReductionDuration * 1000, Timeout.
                Infinite);
        } else {

            foreach (var ip in UDPAdressesCount.Keys) {
                if (UDPAdressesCount [ip] >
                    UDPMaximumTrafficPerIPToShutDownPerSecond) {
                    Console.WriteLine ($"Byl detekovan hranicni pocet {
                        UDPMaximumTrafficPerIPToShutDownPerSecond} UDP paketu
                        z jedne adresy za sekundu." +
                                    $"Adresa { ip.ToString ()} zablokovana
                                        na {
                                        UDPMaximumTrafficPerIPToShutDownDuration
                                        } sekund.");
                    ExecuteCommand($"ebtables -A FORWARD -p IPv4 --ip-source
                        {ip.ToString ()} -j DROP");
                    var autoEvent = new AutoResetEvent (false);
                    var timer = new Timer (
                        MaximumTrafficPerIPToShutDownPerSecond_Timer, ip.
                        ToString (), UDPMaximumTrafficPerIPToShutDownDuration
                        * 1000, Timeout.Infinite);
                }
            }
        }

        UDPPacketCount = 0;
        UDPAdressesCount.Clear ();
    }

    static void ICMPTimer_Timer (Object stateInfo)
    {
        var currentTraffic = ICMPPacketCount;

        Console.WriteLine ($"ICMPPacketCount/s: {currentTraffic}");
```

```
if (currentTraffic > ICMPMaximumTrafficToShutDownPerSecond) {
    Console.WriteLine ($"Byl detekovan maximalni pocet {
        ICMPMaximumTrafficToShutDownPerSecond} ICMP paketu za sekundu
        . " +
                    $"Sit bude na {
                        ICMPMaximumTrafficToShutDownDuration} sekund
                         nedostupna - Podezreni na probihajici DDoS
                        utok");
    completeShutDownICMP = true;
    ExecuteCommand($"ebtables -I FORWARD -p IPv4 --ip-protocol icmp
        -j DROP");
    var autoEvent = new AutoResetEvent (false);
    var timer = new Timer (
        MaximumTrafficICMPToShutDownPerSecond_Timer, autoEvent,
        ICMPMaximumTrafficToShutDownDuration * 1000, Timeout.Infinite
        );
} else if (currentTraffic > ICMPMaximumTrafficPerSecond &&
    completeShutDownICMP == false) {
    var ruleName = _random.Next (0, 999999).ToString ();
    Console.WriteLine ($"Byl detekovan hranicni pocet {
        ICMPMaximumTrafficPerSecond} ICMP paketu za sekundu. " +
                    $"ICMP provoz bude omezen na {
                        ICMPMaximumTrafficPacketPerSecond} paketu za
                         sekundu na dobu {
                        ICMPMaximumTrafficReductionDuration} sekund.
                        ");
    ExecuteCommand($"ebtables -N {ruleName};" +
                   $"ebtables -A FORWARD -p IPv4 --ip-protocol icmp -
                       j {ruleName};" +
                   $"ebtables -A {ruleName} -p IPv4 --limit {
                       ICMPMaximumTrafficPacketPerSecond}/second --
                       limit-burst 3 -j ACCEPT;" +
                   $"ebtables -A {ruleName} -j DROP");
    var timer = new Timer (MaximumTrafficICMPPerSecond_Timer,
        ruleName, ICMPMaximumTrafficReductionDuration * 1000, Timeout
        .Infinite);
} else {
    foreach (var ip in ICMPAdressesCount.Keys) {
```

```
            if (ICMPAdressesCount [ip] >
                ICMPMaximumTrafficPerIPToShutDownPerSecond) {
                Console.WriteLine ($"Byl detekovan hranicni pocet {
                    ICMPMaximumTrafficPerIPToShutDownPerSecond} ICMP
                    paketu z jedne adresy za sekundu." +
                                $"Adresa { ip.ToString ()} zablokovana
                                    na {
                                    ICMPMaximumTrafficPerIPToShutDownDuration
                                    } sekund.");
                ExecuteCommand($"ebtables -A FORWARD -p IPv4 --ip-source
                    {ip.ToString ()} -j DROP");
                var autoEvent = new AutoResetEvent (false);
                var timer = new Timer (
                    MaximumTrafficPerIPToShutDownPerSecond_Timer, ip.
                    ToString (), ICMPMaximumTrafficPerIPToShutDownDuration
                     * 1000, Timeout.Infinite);
            }
        }
    }

    ICMPPacketCount = 0;
    ICMPAdressesCount.Clear ();
}

static void ICMPV6Timer_Timer (Object stateInfo)
{
    var currentTraffic = ICMPV6PacketCount;

    Console.WriteLine ($"ICMPV6PacketCount/s: {currentTraffic}");

    if (currentTraffic > ICMPV6MaximumTrafficToShutDownPerSecond) {
        Console.WriteLine ($"Byl detekovan maximalni pocet {
            ICMPV6MaximumTrafficToShutDownPerSecond} ICMPv6 paketu za
            sekundu. " +
                        $"Sit bude na {
                            ICMPV6MaximumTrafficToShutDownDuration}
                            sekund nedostupna - Podezreni na probihajici
                             DDoS utok");
        completeShutDownICMP = true;
```

```
            ExecuteCommand($"ebtables -I FORWARD -p IPv6 --ip6-protocol ipv6
                -icmp -j DROP");
            var autoEvent = new AutoResetEvent (false);
            var timer = new Timer (
                MaximumTrafficICMPV6ToShutDownPerSecond_Timer, autoEvent,
                ICMPV6MaximumTrafficToShutDownDuration * 1000, Timeout.
                Infinite);
        }
        else if (currentTraffic > ICMPV6MaximumTrafficPerSecond &&
            completeShutDownICMPV6 == false) {
            var ruleName = _random.Next (0, 999999).ToString ();
            Console.WriteLine ($"Byl detekovan hranicni pocet {
                ICMPV6MaximumTrafficPerSecond} ICMPv6 paketu za sekundu. " +
                            $"ICMP provoz bude omezen na {
                                ICMPV6MaximumTrafficPacketPerSecond} paketu
                                za sekundu na dobu {
                                ICMPV6MaximumTrafficReductionDuration}
                                sekund.");
            ExecuteCommand($"ebtables -N {ruleName};" +
                           $"ebtables -A FORWARD -p IPv6 --ip6-protocol ipv6-
                               icmp -j {ruleName};" +
                           $"ebtables -A {ruleName} -p IPv6 --limit {
                               ICMPV6MaximumTrafficPacketPerSecond}/second --
                               limit-burst 3 -j ACCEPT;" +
                           $"ebtables -A {ruleName} -j DROP");
            var timer = new Timer (MaximumTrafficICMPV6PerSecond_Timer,
                ruleName, ICMPV6MaximumTrafficReductionDuration * 1000,
                Timeout.Infinite);
        }
    }

    static void MaximumTrafficICMPPerSecond_Timer (Object stateInfo)
    {
        var ruleName = stateInfo.ToString ();
        Console.WriteLine ($"Provoz ICMP po omezeni obnoven.");
        ExecuteCommand($"ebtables -D FORWARD -p IPv4 --ip-protocol icmp
            -j {ruleName};" +
                       $"ebtables -D {ruleName} -p IPv4 --limit {
                           ICMPMaximumTrafficPacketPerSecond}/second --
```

```
                    limit-burst 3 -j ACCEPT;" +
                $"ebtables -D {ruleName} -j DROP;" +
                $"ebtables -X {ruleName}");
}

static void MaximumTrafficICMPV6PerSecond_Timer (Object stateInfo)
{
var ruleName = stateInfo.ToString ();
Console.WriteLine ($"Provoz ICMPv6 po omezeni obnoven.");
ExecuteCommand ($"ebtables -D FORWARD -p IPv6 --ip6-protocol ipv6-
    icmp -j {ruleName};" +
            $"ebtables -D {ruleName} -p IPv6 --limit {
                ICMPV6MaximumTrafficPacketPerSecond}/second --
                limit-burst 3 -j ACCEPT;" +
            $"ebtables -D {ruleName} -j DROP;" +
            $"ebtables -X {ruleName}");
}

static void MaximumTrafficUDPPerSecond_Timer (Object stateInfo)
{
    var ruleName = stateInfo.ToString ();
    Console.WriteLine ($"Provoz UDP po omezeni obnoven.");
    ExecuteCommand($"ebtables -D FORWARD -p IPv4 --ip-protocol udp -
        j {ruleName};" +
                $"ebtables -D {ruleName} -p IPv4 --limit {
                    ICMPMaximumTrafficPacketPerSecond}/second --
                    limit-burst 3 -j ACCEPT;" +
                $"ebtables -D {ruleName} -j DROP;" +
                $"ebtables -X {ruleName};");
}

static void MaximumTrafficTCPPerSecond_Timer (Object stateInfo)
{
    var ruleName = stateInfo.ToString ();
    Console.WriteLine ($"Provoz TCP po omezeni obnoven.");
    ExecuteCommand($"ebtables -D FORWARD -p IPv4 --ip-protocol tcp -
        j {ruleName};" +
                $"ebtables -D {ruleName} -p IPv4 --limit {
                    ICMPMaximumTrafficPacketPerSecond}/second --
```

```
                    limit-burst 3 -j ACCEPT;" +
                $"ebtables -D {ruleName} -j DROP;" +
                $"ebtables -X {ruleName};");
}

static void MaximumTrafficTCPToShutDownPerSecond_Timer (Object
    stateInfo)
{
    completeShutDownTCP = false;
    Console.WriteLine ($"TCP provoz obnoven.");
    ExecuteCommand($"ebtables -D FORWARD -p IPv4 --ip-protocol tcp -
        j DROP");
}

static void MaximumTrafficUDPToShutDownPerSecond_Timer (Object
    stateInfo)
{
    completeShutDownUDP = false;
    Console.WriteLine ($"UDP provoz obnoven.");
    ExecuteCommand($"ebtables -D FORWARD -p IPv4 --ip-protocol udp -
        j DROP");
}

static void MaximumTrafficICMPToShutDownPerSecond_Timer (Object
    stateInfo)
{
    completeShutDownICMP = false;
    Console.WriteLine ($"ICMP provoz obnoven.");
    ExecuteCommand($"ebtables -D FORWARD -p IPv4 --ip-protocol icmp
        -j DROP");
}

static void MaximumTrafficICMPV6ToShutDownPerSecond_Timer (Object
    stateInfo)
 {
    completeShutDownICMPV6 = false;
    Console.WriteLine ($"ICMPv6 provoz obnoven.");
    ExecuteCommand ($"ebtables -D FORWARD -p IPv6 --ip6-protocol
        icmp -j DROP");
```

```
 }

static void MaximumTrafficPerIPToShutDownPerSecond_Timer (Object
    stateInfo)
{
    Console.WriteLine ($"IP adresa {stateInfo.ToString ()} obnovena.
        ");
    ExecuteCommand($"ebtables -D FORWARD -p IPv4 --ip-source {
        stateInfo.ToString ()} -j DROP");
}

private static void Init (string [] args)
{
    TCPMaximumTrafficPerSecond = Convert.ToInt32 (args [0]);
    TCPMaximumTrafficPacketPerSecond = Convert.ToInt32 (args [1]);
    TCPMaximumTrafficReductionDuration = Convert.ToInt32 (args [2]);
    TCPMaximumTrafficToShutDownPerSecond = Convert.ToInt32 (args
        [3]);
    TCPMaximumTrafficToShutDownDuration = Convert.ToInt32 (args [4])
        ;
    TCPMaximumTrafficPerIPToShutDownPerSecond = Convert.ToInt32 (
        args [5]);
    TCPMaximumTrafficPerIPToShutDownDuration = Convert.ToInt32 (args
         [6]);

    UDPMaximumTrafficPerSecond = Convert.ToInt32 (args [7]);
    UDPMaximumTrafficPacketPerSecond = Convert.ToInt32 (args [8]);
    UDPMaximumTrafficReductionDuration = Convert.ToInt32 (args [9]);
    UDPMaximumTrafficToShutDownPerSecond = Convert.ToInt32 (args
        [10]);
    UDPMaximumTrafficToShutDownDuration = Convert.ToInt32 (args
        [11]);
    UDPMaximumTrafficPerIPToShutDownPerSecond = Convert.ToInt32 (
        args [12]);
    UDPMaximumTrafficPerIPToShutDownDuration = Convert.ToInt32 (args
         [13]);

    ICMPMaximumTrafficPerSecond = Convert.ToInt32 (args [14]);
    ICMPMaximumTrafficPacketPerSecond = Convert.ToInt32 (args [15]);
```

```
            ICMPMaximumTrafficReductionDuration = Convert.ToInt32 (args
                [16]);
            ICMPMaximumTrafficToShutDownPerSecond = Convert.ToInt32 (args
                [17]);
            ICMPMaximumTrafficToShutDownDuration = Convert.ToInt32 (args
                [18]);
            ICMPMaximumTrafficPerIPToShutDownPerSecond = Convert.ToInt32 (
                args [19]);
            ICMPMaximumTrafficPerIPToShutDownDuration = Convert.ToInt32 (
                args [20]);

            ICMPV6MaximumTrafficPerSecond = Convert.ToInt32 (args [21]);
            ICMPV6MaximumTrafficPacketPerSecond = Convert.ToInt32 (args
                [22]);
            ICMPV6MaximumTrafficReductionDuration = Convert.ToInt32 (args
                [23]);
            ICMPV6MaximumTrafficToShutDownPerSecond = Convert.ToInt32 (args
                [24]);
            ICMPV6MaximumTrafficToShutDownDuration = Convert.ToInt32 (args
                [25]);
        }

         private static void ExecuteCommand(string command)
        {
            Process proc = new Process ();

            proc.StartInfo.FileName = "/bin/bash";
            proc.StartInfo.Arguments = "-c \" " + command + " \"";
            proc.StartInfo.UseShellExecute = false;
            proc.StartInfo.RedirectStandardOutput = true;
            proc.Start ();

        }
    }
}
```

Výpis 13: Anti-DDoS program psaný v jazyce C# .NET Core na Ubuntu 14.04